UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

Departamento de Arquitectura de Computadores y Automática

TESIS DOCTORAL

Ejecución eficiente de códigos Monte Carlo en infraestructuras de Grid

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Manuel Rodríguez-Pascual

Directores

Rafel Mayo-García
Ignacio Martín Llorente

Madrid, 2012

© Manuel Rodríguez-Pascual, 2012

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMATICA
Departamento de Arquitectura de Computadores y Automática

Ejecución eficiente de códigos Monte Carlo en infraestructuras Grid

Memoria que presenta para optar al título de Doctor en Informática

Manuel Rodríguez-Pascual

Dirigida por los doctores

Dr. Rafael Mayo-García
Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas
Facultad de Ciencias Físicas, Universidad Complutense de Madrid

Dr. Ignacio Martín Llorente
Facultad de Informática, Universidad Complutense de Madrid

Madrid, 2011

Documento maquetado con TEXIS v.1.0+.

Ejecución eficiente de códigos Monte Carlo en infraestructuras Grid

TESIS DOCTORAL

Manuel Rodríguez-Pascual

Madrid, 2011

Departamento de Arquitectura de Computadores y Automática

Universidad Complutense de Madrid

Civilization advances by extending the number of important operations which we can perform without thinking of them
Alfred North Whitehead, An Introduction to Mathematics*, 1911*

Índice

Índice de figuras

Índice de Tablas

Agradecimientos

Mi primer agradecimiento es para los co-directores de esta tesis, **Ignacio Martín Llorente** y **Rafael Mayo.** Desde que empecé a colaborar con su departamento en las horas libres que me dejaba la carrera hasta el momento en que puse el último punto de esta tesis, Nacho ha sido la persona que ha marcado mi trayectoria en el mundo de la investigación. Y cuando sugirió que solicitara una beca en el CIEMAT -mi trabajo durante los últimos 4 años- me dejó en las mejores manos posibles, las de **Rafael Mayo**. Con su paciencia, sus consejos y saber hacer, Rafa ha sido mi guia en periodo en que he ido aprendiendo de qué va todo esto de la investigación.

En La Complutense dejé atrás a muchos compañeros y amigos, a los que que por suerte aun sigo viendo con asiduidad. **Tino Vázquez**, **Javi Fontán** y **Jose Luis Vázquez-Poletti** me descubrieron la parte más humana de la investigación y enseñaron -entre otro montón de cosas- que el compaginar el trabajo y la diversión no es sólo posible, sino necesario.

Al llegar al CIEMAT tuve una gran acogida: la que me brindó mi compañero de despacho **Antonio Juan Rubio Montero, Toñete**. Juntos hemos comido codillo en los Alpes, arepas en el Caribe y ñandú en la Patagonia. Pero también hemos pasado cientos de horas en el despacho sacando adelante trabajo en plazos imposibles, luchando por encontrar la frase perfecta para un artículo mientras fuera las horas pasaban más rápido de lo que nadie puede imaginar.

Esther Montes, **Jorge Blanco** e **Iván Casas** marcan la diferencia entre un trabajo rutinario y uno al que llegas con una sonrisa en la boca -de manera figurada, por desgracia no tengo el mejor de los despertares. Día a día, han pasado de ser perfectos desconocidos a alguien a quien echo de menos si desaparecen una mañana y a quién estoy deseando ver al volver de vacaciones o un viaje. Ojalá haya suerte y podamos seguir trabajando juntos muchos años más.

Volviendo a la parte académica, quiero agradecer al **Dr. Paco Castejón** sus explicaciones sobre física del plasma. Así mismo, ha sido la continua colaboración entre su departamento y mi grupo de trabajo la que me ha permitido participar en multitud de proyectos de fusión. Sin su entusiasmo y creatividad, este periodo en el CIEMAT hubiera sido completamente distinto

para mi: seguramente con bastantes menos sobresaltos, pero sin duda más monótono y aburrido.

El **Dr. Bruce Scott** me acogió en el Instituto Marx Planck de Munich durante los tres meses que duró mi estancia. Ha sido un placer trabajar con él, y comprobar como alguien de reconocido prestigio en su campo puede ser además alguien cercano y atento con un becario recién llegado y algo perdido.

Por diferentes motivos, durante el otoño del 2010 pasé casi dos meses en Chile, repartidos entre Chillán, Santiago y Valparaíso. El buen hacer de su gente, su amabilidad y capacidad de trabajo, consiguieron que a la semana de estar ahí sintiera que llevaba toda una vida. El equipo del **Dr. Salinas** (especialmente **Raquel Pezoa** y **Paola Arce**) contribuyó decisivamente a que guarde un grato recuerdo de esta estancia. Desde aquí les envío mi más sincero agradecimiento.

Tanto el código fuente de FAFNER2 como los datos de muestra que se emplearon como ejemplo de ejecución me han sido facilitados por el **Dr. José Guasp**. Así mismo, sus consejos y sugerencias fueron una gran orientación a la hora de entender y modificar el código de la aplicación. ISDEP fue integrado con la ayuda de **Andrés Bustos**, quien además me enseñó la única batamanta que he tenido oportunidad de ver en persona.

En cualquier caso, este trabajo sólo ha sido posible gracias al trabajo diario del **personal dedicado al mantenimiento y mejora de las instalaciones del Centro de Cálculo de Madrid del CIEMAT**, donde se albergan la mayoría de los equipos utilizados. Para ellos, y especialmente para el Director de la División de TIC **Fernando Blanco**, mi más sincero agradecimiento.

Es difícil nombrar a todas las personas que han hecho que este periodo de investigación haya sido una de las estapas de mi vida de la que mejor recuerdo guardo, así que intentaré resumirlo: si hemos compartido el *coffee-break* de algún congreso, una merecida cerveza después de 8 horas de conferencias o aguantado juntos el retraso de un vuelo, ten presente que has puesto un granito de arena en este trabajo que ahora estás leyendo.

Pero no todo en esta vida es trabajo, y mucha de la gente que ha contribuido decisivamente a esta tesis ni siquiera sabe exactamente a qué me dedico. El primero de ellos es **Beny**, compañero de piso y aventuras los últimos dos años y espero que unos cuantos más. Así mismo, todos los habitantes de La Duquasa presentes y pasados -**Crisis, Ruth, Mery, Belén, Peggy Sue**- han conseguido convertir el llegar a mi salón en un acontecimiento que sabes como empieza pero casi nunca cómo acabará. Sin esta magnífica compañía, las temporadas de stress y vida "de casa al trabajo y del trabajo a casa" hubieran sido mucho más difíciles de sobrellevar.

Primero con **Dako** y ahora con **Barhoppers**, el tocar en bandas de música me ha dado algunos de los mejores momentos de estos últimos años.

Para mi, el llegar agotado mentalmente del trabajo y tener un ensayo en que el desconectar es algo que cambiaría por muy pocas cosas en el mundo. Y para eso tengo suerte de contar con grandes músicos y amigos, además de los vecinos con más paciencia del mundo.

Que hayan tardado en aparecer no significa que no sean importantes **mis compañeros de facultad**. Sois demasiados para que parezcáis aquí todos, lo siento! Así que sólo nombraré a **Lituero** y todos os daréis por incluidos, imagino.

Cuando acabé segundo de bachiller -es decir, el curso previo a la universidad- mi hermano **Miguel** me dijo "si vas a estudiar una ingeniería disfruta este verano, son las últimas vacaciones de verdad que vas a tener en tu vida", una frase ampliamente repetida en tono jocoso por mis padres **Aurelio** y **Natividad** cada vez que desde entonces me he quejado por exceso de trabajo, exámenes o tener que quedarme en casa un sábado noche preparando alguna entrega. Vayan para ellos tres estas últimas líneas y la constatación de que, por desgracia, tenían razón.

Acerca de este documento

Esta Tesis Doctoral se presenta como un trabajo original de investigación, de acuerdo con el punto 4.1 de la Normativa de desarrollo del capítulo V del Real Decreto 1393/2007 del 30 de octubre, por el que se establece la ordenación de las enseñanzas universitarias oficiales (Aprobado en Consejo de Gobierno con fecha 14 de octubre del 2008 y publicado en el BOUC con fecha de 20 de noviembre del 2008).

Por otro lado, este trabajo de Tesis Doctoral opta a la obtención de la mención "Doctor Europeo" según el epígrafe 7.2 de la misma normativa. De acuerdo con los requisitos establecidos en el punto 7.2.1.1-c de la normativa, se incluyen las partes indicadas -el resumen y el apartado de principales aportaciones y trabajo futuro- en inglés, idioma oficial de la Unión Europea distinto del castellano.

About this document

This PhD thesis is presented as an original research work, acording to section 4.1 of *Normativa de desarrollo del capítulo V del Real Decreto 1393/2007, del 30 de octubre, por el que se establece la ordenación de las enseñanzas universitarias oficiales* (approved by the Consejo de Gobierno as of October 14th 2008 and published in the BOUC as of November 20th 2008).

Additionally, this PhD. thesis is presented as a candidate to the obtention of the "Doctor Europeus" mention, following the indications of epigraph 7.2 from the same normative. According to the requisites stablished on epigraph 7.2.1.1-c of this normative, the required parts of this work -the summary and the Section containing the main contributions and future work- are presented in English, an official European Union language different than Spanish.

Resumen

Entre los diferentes aspectos que rodean a la computación Grid, esta tesis se centra en la ejecución eficiente de códigos de Monte Carlo.

Hay determinados problemas en los que, por su complejidad o tamaño, es muy difícil o imposible obtener una solución exacta. Esto no hace que sean dejados de lado; por el contrario, se han creado multitud de alternativas que permiten aproximaciones más o menos precisas al resultado teórico empleando una fracción de los recursos computacionales, lo que permite su uso en entornos reales.

Entre estas alternativas se encuentra el llamado *método de Monte Carlo* (MC). Este método se apoya en el hecho de que algunos fenómenos complejos pueden ser modelados mediante la ejecución de diferentes simulaciones independientes, empleando números aleatorios para obtener los resultados. Gracias a esta particularidad, los códigos de Monte Carlo son ampliamente empleados en diferentes áreas del conocimiento. Además, su modularidad simplifica la ejecución en entornos distribuidos, donde la asignación de tareas a los recursos computacionales y la sincronización de las mismas dista de ser trivial.

Debido a la alta demanda de recursos en muchas aplicaciones científicas -los códigos de MC entre ellas- la Computación Grid se ha erigido como una de las plataformas donde abordar problemas cada vez mayores y más ambiciosos. Además, ha permitido el acceso de la comunidad científica a grandes recursos computacionales, algo especialmente útil dado que el empleo de supercomputadores todavía dista de ser un recurso habitual en determinados entornos.

Debido a las particularidades de este tipo de infraestructuras, la ejecución eficiente de aplicaciones distribuidas está lejos de ser trivial. A pesar del enorme trabajo realizado por la comunidad científica en este área (centrado en diferentes conceptos de "infraestructura Grid", la aplicación a ejecutar o la información disponible) todavía falta una herramienta que permita ejecutar aplicaciones basadas en Monte Carlo en la Grid, que aproveche al máximo su flexibilidad y modularidad para obtener el mayor rendimiento posible.

Para llevar a cabo esta tarea se ha seguido una estrategia innovadora. Aquí los códigos de Monte Carlo, los recursos disponibles y las infraestruc-

turas Grid han sido caracterizados con la mayor información posible, de modo que ésta se pueda emplear para hacer una planificación precisa. Esto supone un avance respecto a las herramientas previamente existentes, dado que el empleo de un volumen mucho mayor de información en el proceso de planificación permite llevar a cabo políticas más eficientes tanto en tiempo de ejecución de la aplicación como en el empleo de recursos.

Además se ha creado un nuevo algoritmo de *self-scheduling* llamado DyTSS (acrónimo de Dynamic Trapezoidal Self Scheduling, Auto-planificación Trapezoidal Dinámica), que utiliza esta información para decidir dónde ejecutar cada tarea y el tamaño de la misma.

A pesar de que la gran mayoría de los trabajos de *scheduling* existentes se centran en el uso de simuladores o entornos controlados, en la presente investigación estas propuestas han sido implementadas y empleadas en infraestructuras Grid en producción. Para ello se muestra el funcionamiento de Montera (acrónimo de MONTE Carlo RApido), un *framework* desarrollado durante el transcurso de la presente tesis. En Montera se han incluido mecanismos que obtienen la información necesaria para los modelos antes mencionados, y se han implementado diferentes algoritmos de planificación entre los que se incluye DyTSS.

En resumen, en esta tesis se han ejecutado los siguientes pasos: caracterización del entorno Grid orientada a códigos de MC; *profiling* de los códigos de MC; diseño de un algoritmo que emplea estos modelos para ejecutar de manera eficiente las aplicaciones; e implementación de todo lo anterior en una única aplicación software.

Summary

Among the different aspects that compound Grid Computing this thesis is focused on the efficient execution of Monte Carlo codes.

Monte Carlo (MC) codes rely on the assumption that the behavior of complex phenomena can often be modeled by the execution of multiple simple simulations that employ random or pseudo-random numbers to compute their results. Thanks to this particularity, MC codes are widely used in various fields of computing simulations, solving problems where it is unfeasible or impossible to compute an exact result with a deterministic algorithm. The modularity of MC codes also simplifies the execution of distributed environments, where task distribution and synchronization is not trivial.

Because of the high demand many scientific applications place on resources -MC codes among them- Grid computing has emerged as a powerful platform for facing new and more ambitious problems. Grid computing has enabled the scientific community to have easier access to large computing resources, something specially interesting nowadays that the employment of supercomputers is still far from being a commodity.

Due to the particularities of this kind of infrastructure, the efficient execution of distributed applications is far from trivial. Although an enormous effort has been put into this area by the scientific community (with many different approaches focusing on different concepts of "Grid Infrastructure", the particular characteristics of the application to be executed or the available and published information) there is still the lack of a specific tool for executing Monte Carlo based applications on the Grid, taking the most of its flexibility and modularity to obtaining the highest performance.

To carry on this task an innovative strategy has been followed. Here, Monte Carlo codes, available resources and Grid Infrastructures have been characterized with as much information as possible, employing it to perform a precise scheduling. This represents an advantage respect the previously existing tools: the employment of a bigger volume of information in the scheduling process leads to more efficient policies both in execution time and in resource usage.

Also, a new self-scheduling algorithm called DyTSS (acronim for Dynamic Trapezoidal Self-Scheduling) has been created. It employes this informa-

tion to decide where to submit each task and its size.

Even though most of the previous works on the area use simulators or controlled environments, in the current research these proposals have been implemented and employed on Grid infrastructures in production status. To do so Montera (acronym for *MONTE carlo RApido*, Fast Monte Carlo in spanish), a framework for efficient executions of Monte Carlo codes has been developed. Montera includes mechanisms that obtain the necessary information to the aforementioned models, and different scheduling algorithms -including DyTSS- have been implemented.

In summary, this work describes a proposal for optimizing the execution of MC codes on the Grid. First, a characterization of both MC codes and Grid infrastructures is detailed. Then, an algorithm that employs this information to split the desired simulation into tasks of different sizes and submit them to the Grid is explained. Finally, Montera, a framework that implements the proposed characterizations and the algorithm, is presented.

Capítulo 1

Introducción

El área de la informática denominada *computación distribuida* se centra en la ejecución de aplicaciones empleando más de un recurso computacional. Dichos recursos se comunican mediante una red de datos, y cooperan para obtener el mayor rendimiento posible.

Por su parte, las aplicaciones a ser ejecutadas en entornos distribuidos tienen que poder ser divididas en fragmentos que se ejecutarán en diferentes recursos. En general, estas aplicaciones tienen que tratar con entornos heterogéneos, redes de diferentes latencias, y fallos impredecibles en la red o las computadoras. Todos estos factores han de ser tenidos en cuenta a la hora de diseñarlas, de manera que la influencia de estos factores en el resultado final sea lo más limitada posible.

Según el diseño de la aplicación y el tipo de problema a resolver, las aplicaciones pueden ser divididas en *alto rendimiento* y *alta productividad*. Las primeras se centran en la reducción del tiempo de ejecución de una aplicación mediante la colaboración de múltiples unidades de proceso. Las segundas, en la ejecución del máximo número posible de tareas por unidad de tiempo.

Como regla general, los sistemas de alto rendimiento ejecutan aplicaciones fuertemente acopladas, esto es, con muchas dependencias de datos entre las tareas que se ejecutan en los distintos nodos. Por tanto, es conveniente ejecutarlas en un único *site* equipado con interconexiones de baja latencia. En el otro extremo están los sistemas de alta productividad, que ejecutan trabajos secuenciales independientes. Estos trabajos pueden ser planificados en muchos recursos computacionales, e incluso en diferentes dominios administrativos.

Las tareas de alto rendimiento se caracterizan por necesitar una gran cantidad de recursos computacionales (capacidad de cómputo o consumo de memoria) durante períodos relativamente cortos de tiempo. Por otro lado, las aplicaciones de alta productividad, al estar compuestas por pequeñas tareas independientes, no necesitan una gran cantidad de recursos en un momento

y *site* dado, sino que su ejecución puede ser distribuida o fragmentada según las necesidades del momento. Como se podrá deducir de las explicaciones siguientes, este es el tipo de tarea más adecuado para la ejecución en Grid.

El objetivo de la llamada Computación Grid es *coordinar la compartición de recursos y resolución de problemas en organizaciones virtuales dinámicas y multi-institucionales*. De una manera menos académica, puede decirse que la computación Grid permite a usuarios de diferentes dominios administrativos (centros de investigación, universidades, etc.) el compartir sus equipos, de manera que los momentos en los que un recurso está disponible en una determinada institución puede ser utilizado por un trabajador de otra. De este modo se alcanza el llamado *supercomputador más extenso del mundo*, donde los participantes tienen a su disposición equipos de diferentes instituciones, países, y en algunos casos incluso continentes. Por supuesto esta compartición dista de ser trivial, y crea una serie de retos técnicos y administrativos que es preciso resolver.

El hecho de que los recursos pertenezcan a diferentes administraciones provoca que el usuario no tenga un control absoluto de los mismos. Por el contrario, su uso está supeditado a su disponibilidad y autorización del responsable de los mismos, y en general tendrán una configuración y política de seguridad diferentes a las de las infraestructuras propias.

Como consecuencia de estas características, las redes Grid son entornos muy dinámicos en término de disponibilidad de recursos. Las aplicaciones destinadas a correr en este tipo de infraestructura deben por tanto ser capaces de adaptarse a un entorno cambiante, aprovechando los nuevos recursos que aparecen y recuperándose de fallos en aquellos donde se está ejecutando. Esto favorece a las aplicaciones de alta productividad compuestas por tareas independientes, donde los problemas en la ejecución de una de ellas no comprometen el resultado de las demás. Y dentro de este tipo de aplicaciones, los códigos Monte Carlo ocupan un lugar destacado.

El método de Monte Carlo se basa en el hecho de que el comportamiento de fenómenos complejos puede ser modelado mediante la ejecución de múltiples simulaciones simples, empleado números aleatorios o pseudoaleatorios para computar los resultados.

Gracias a esta particularidad el método de Monte Carlo es ampliamente empleado en diferentes áreas del conocimiento, resolviendo problemas donde el obtener un resultado exacto con un algoritmo determinista es imposible o inabarcable por su complejidad. Además, su modularidad simplifica su ejecución en entornos distribuidos, al poder dividir una aplicación en el número de tareas deseadas sin que eso suponga una pérdida destacable del rendimiento o diferencia en los resultados obtenidos.

La ejecución de códigos Monte Carlo en entornos distribuidos se divide en tres secciones diferenciadas. En la primera, llevada a cabo en el recurso local, se toman las decisiones correspondientes a la ejcecución de las tareas.

Para ello se determina el número y situación de los recursos a emplear, así como las simulaciones que se llevarán a cabo de cada uno. A continuación las tareas son ejecutadas en el recurso remoto y llevan a cabo el número deseado de simulaciones. Por último los resultados de estas ejecuciones son devueltos al recurso local, donde son procesados para obtener el resultado final.

Las aplicaciones que siguen esta estructura permiten ser ejecutadas en infraestructuras Grid de una manera sencilla y eficiente. Por un lado, la ausencia total de comunicación entre las distintas tareas hace que no existan limitaciones en su escalabilidad, más allá del número de recursos disponibles. Por otro, la sobrecarga producida con el aumento de tareas se limita a la copia de los datos de entrada y salida a cada uno de los recursos remotos, y existe multitud de soluciones que consiguen minimizar y/o paralelizar esta transferencia de datos.

La presente tesis doctoral se centra en la primera fase de la ejecución de Monte Carlo en Grid: decidir el número y tamaño de tareas a ejecutar en cada uno de los recursos disponibles. Este campo del conocimiento es denominado planificación de tareas (o *scheduling* utilizando un anglicismo) y constituye un amplísimo campo de estudio donde pueden seguirse multitud de enfoques.

Aquí se presenta un nuevo algoritmo de *scheduling* llamado DyTSS (*Dynamic Trapezoidal Self Scheduling*, Auto Planificación Trapezoidal Dinámica). Se trata de una versión modificada del algoritmo GTSS (*Grid Trapezoidal Self Scheduling*, Auto Planificación Trapezoidal en Grid) adaptado para ejecutar códigos de Monte Carlo en infraestructuras extremadamente dinámicas. Este algoritmo es presentado desde un punto de vista teórico, junto a la información que es necesario obtener de la infraestructura Grid y la aplicación para poder aplicarlo. El siguiente paso consistió entonces en la creación de un *framework* que implementa este algoritmo junto a las herramientas auxiliares para poder ejecutarlo, y se ha demostrado su utilidad en entornos en producción. Dicho *framework* ha sido bautizado Montera (acrónimo de *MONTE carlo RApido*) por su gran eficiencia ejecutando este tipo de aplicaciones.

1.1. Objetivos

El objetivo final de esta tesis es la creación de un *framework* que permita una ejecución eficiente, robusta y sencilla de códigos basados en el paradigma de Monte Carlo en infraestructuras Grid. Para cumplir estos objetivos, al *framework* desarrollado se le exigirán una serie de requisitos.

El primero es que el tiempo de ejecución de un código de Monte Carlo sea inferior al obtenido con el resto de las opciones existentes. Este es el requisito más importante, ya que de no cumplirse, este trabajo carecería de sentido.

Además, el *framework* desarrollado deberá ser resistente a fallos, tanto de las propias tareas como del hardware. Ambos son problemas típicos en la ejecución de tareas en Grid -como será explicado con detalle en la siguiente sección- por lo que cualquier herramienta destinada a ser empleada en entornos de producción ha de poder lidiar con ellos.

Por último, se espera que el *framework* sea lo bastante sencillo de usar para que un usuario novel pueda aprender a dominarlo en poco tiempo, y a la vez lo bastante versátil para que un usuario avanzado sea capaz de configurarlo de acuerdo a sus necesidades.

1.2. Estructura

A continuación se define la estructura y organización de la presente Tesis Doctoral.

El primer capítulo, *Computación en Grid*, explica los conceptos claves de la computación en Grid, centrándose en la computación científica de alto rendimiento.

Como se verá, el concepto de infraestructura Grid no es único, y existen distintas interpretaciones y arquitecturas según la definición que se proporcione. Por eso este apartado es fundamental, ya que sienta las bases sobre las que se trabajará después. Para ello también incorpora una descripción de las herramientas utilizadas, junto con la razón de haber elegido esas en concreto dentro del gran abanico disponible.

El siguiente apartado, *Códigos de Monte Carlo*, proporciona una visión de alto nivel del método de Monte Carlo, su implementación en los llamados "códigos de Monte Carlo" y su importancia en la computación.

Para ello, se ha decidido incluir una breve historia del método de Monte Carlo desde su creación en la Segunda Guerra Mundial hasta la actualidad y su futuro previsible. La intención de esto es evitar el planteamiento de esta Tesis Doctoral como un trabajo de laboratorio aislado de la realidad, explicando su utilidad práctica y cómo se enmarca en una corriente de trabajo que durante décadas ha sido de enorme utilidad práctica sin perder vigencia en su planteamiento. Así mismo se incluye una pequeña proyección del futuro de la computación y el lugar que previsiblemente ocuparan los códigos de Monte Carlo, con la intención de mostrar que seguirán siendo de interés al menos a corto y medio plazo.

Y por último, se detallan las maneras actuales de ejecutar los códigos de Monte Carlo en infraestructuras Grid. Se resaltan los trabajos más representativos en el área hasta el momento y se analizan sus problemas, demostrando así la necesidad de esta nueva propuesta.

La sección sobre *Planificación de iteraciones en Grid* es la que contiene una mayor carga teórica dentro de esta Tesis Doctoral. Su contenido se centra en una técnica de planificación de tareas en Grid llamada "planificación de

iteraciones", abordada desde distintos ángulos. De este modo se describirá el estado del arte y las herramientas que han constituido la base del algoritmo DyTSS, que constituye una de las principales aportaciones de la presente tesis doctoral.

Primero, se hace una introducción a la planificación de iteraciones. Se definen los conceptos que es necesario conocer, el origen de la planificación de iteraciones, y su validez en los entornos computacionales actuales. Así mismo, se detalla la manera de extrapolar este tipo de algoritmos a los códigos de Monte Carlo, de manera que pueda decidirse de una manera automática cuántos *samples* han de simularse en cada tarea para minimizar su tiempo total de ejecución. Posteriormente se detallan los distintos algoritmos de planificación de iteraciones, así como las ventajas e inconvenientes de cada uno.

Por último se presenta y analiza en profundidad el algoritmo DyTSS, una novedosa propuesta de *scheduling* que sigue diseñado específicamente para adaptarse a las particularidades de los códigos de Monte Carlo obteniendo así un gran rendimiento. Este algoritmo supone la principal aportación intelectual de la presente tesis.

La sección de *Montera* analiza este *framework* en profundidad. Para ello incluye tanto una análisis teórico de los problemas a abordar y la manera de solucionarlos como una descripción de la implementación de dichas soluciones.

Comienza con una introducción acerca de Montera: sus capacidades, en qué se diferencia del resto de planificadores de tareas en Grid y la necesidad de su existencia.

Posteriormente se detalla cómo Montera modela los distintos recursos que tiene a su alcance, para de este modo poder realizar una planificación más precisa. Se explica cómo realiza perfiles de la ejecución de códigos de Monte Carlo, y obtiene información acerca de los recursos que componen una determinada infraestructura Grid para poder calcular de una manera más precisa el rendimiento de los mismos al trabajar con aplicaciones intensivas en CPU (*Central Process Unit*, Unidad de Procesamiento Central).

Por último, se describe la arquitectura de Montera y los distintos métodos de planificación que incorpora, proveyendo así al lector de una explicación del *framework* a bajo nivel.

En la sección *Análisis de las propuestas* se lleva a cabo un análisis cuantitativo de las propuestas de esta Tesis. Su función es demostrar que el trabajo realizado no sólo es válido desde un punto de vista teórico, sino que tiene una utilidad práctica a la hora de ejecutar códigos de Monte Carlo, y la mejora en los tiempos de ejecución obtenidos representa una ventaja real respecto a las soluciones previas.

Para ello se planteará el uso de un pequeño simulador, creado en el transcurso de esta Tesis para comprobar el rendimiento obtenido con distintas

políticas de planificación. Este simulador está incorporado en el *framework* Montera, y es capaz de aprovechar la información obtenida sobre la infraestructura Grid y el código MC en ejecuciones reales previas de Montera para de ese modo poder realizar una predicción precisa. Así, es posible determinar el valor apropiado de ciertos parámetros de los algoritmos de planificación, y diferentes propuestas pueden ser comparadas antes de ejecutar las mediciones en un entorno real. De este modo el tiempo de desarrollo de un nuevo algoritmo se disminuye, a la vez que se reduce la carga de trabajo sobre la infraestructura Grid.

Después, se presenta el banco de pruebas donde se llevó a cabo el análisis de rendimiento. Finalmente se llevó a cabo dicho análisis de rendimiento empleando tres aplicaciones distintas, quedando probado que Montera es una solución válida para llevar a cabo el trabajo para el que se diseñó.

Por último, en la sección de *Principales aportaciones y trabajo futuro* se detalla el trabajo realizado desde un punto de vista cualitativo, explicando las principales contribuciones al conocimiento. Por último se describen posibles mejoras futuras al trabajo, planteando nuevas líneas de investigación en las que aprovechar el conocimiento aquí adquirido para seguir desarrollando software que resulte útil a los investigadores para llevar a cabo su labor de la manera más sencilla y eficiente posible.

Adicionalmente se han incluido tres apéndices relativos a las aplicaciones FAFNER2, ISDEP y FastDEP. En ellos se ha descrito su proceso de portado a Grid y su ejecución en Grid bajo distintos paradigmas, para poder así cuantificar la mejora de rendimiento obtenida gracias a Montera.

Se ha optado por incluir esta información en apéndices ya que, a pesar de resultar de interés para comprender el desarrollo de esta tesis, suponen una aplicación práctica de la investigación propuesta sin aportar más contenidos desde un punto de vista teórico.

Capítulo 2

Computación en Grid

En la presente sección se describirá la computación en Grid, junto a las herramientas utilizadas y las alternativas descartadas. De este modo se pretende proporcionar al lector una visión general de la infraestructura en la que se llevará a cabo el resto del trabajo.

En el caso de la computación Grid este paso es especialmente necesario, dada la diversidad de opiniones y los distintos enfoques que es posible encontrar en la bibliografía especializada. Al contrario que en otras áreas del conocimiento, en el caso de la computación Grid no existe una definición exclusiva y universalmente aceptada, por lo que se es conveniente el introducir los conceptos que se manejan.

En parte por esta falta de homogeneización, y en parte por las diferentes necesidades computacionales en distintos ámbitos, existe una gran cantidad de herramientas distintas para la ejecución de trabajos en Grid. En esta sección de describirán algunos de los más importantes, junto a los motivos por los que se emplearon unas en concreto.

2.1. Introducción

La computación Grid es una de las soluciones más naturales a determinados problemas que se presentan actualmente en la computación de alto rendimiento.

Por un lado, las necesidades de computación de los científicos usualmente no son lineales, sino que oscilan enormemente a lo largo del tiempo. Esto es inherente al ciclo habitual que sigue una parte significativa de los procesos de investigación: decisión del código a utilizar -o creación de uno nuevo-; pruebas y ajustes del mismo; ejecución masiva; análisis de los resultados; y presentación de los mismos en un foro adecuado. Como se puede observar, del tiempo que acabarca una investigación completa sólo una parte del mismo suele requerir grandes capacidades de computación.

Esto plantea un dilema que históricamente se ha resuelto de maneras muy diversas: el alcanzar un equilibrio entre los picos de necesidad computacional de los usuarios y la inversión en infraestructuras. Una deficiente capacidad computacional supone un cuello de botella en las investigaciones, pero el satisfacer las necesidades en los momentos álgidos provoca que las infraestructuras permanezcan infrautilizadas parte de su tiempo, con el desperdicio de recursos que esto supone. Es importante tener en cuenta que cualquier infraestructura informática consume energía por el mero hecho de y tiene una vida limitada antes de quedar obsoleta, por lo que una utilización por debajo de sus capacidades supone además un considerable gasto económico.

En este ámbito, la computación Grid propone una solución a este problema. La idea básica consiste en una compartición de recursos entre centros de investigación, de manera que una parte de su infraestructura esté a disposición de los demás. De este modo, cuando un determinado investigador tiene grandes necesidades computacionales es capaz de utilizar la potencia combinada de decenas de centros, mientras que cuando no lo necesita son otros los que emplean sus recursos. Así, el mencionado equilibrio entre oferta y demanda puntual se alcanza sin necesidad de fuertes inversiones económicas.

Este comportamiento puede apreciarse en la figura 2.1, que representa el tiempo de CPU empleado por las Organizaciones Virtuales de la infraestructura EGI a lo largo del 2010 -excluyendo *High Energy Physics* para aumentar su legibilidad, ya su número de horas es mucho mayor que todos los demás juntos. A pesar de que cada uno tiene una demanda irregular de cálculo, la suma total no experimenta grandes variaciones. Así pues, queda patente que, en determinadas circunstancias, la cooperación de usuarios permite un mejor aprovechamiento de las infraestructuras a la vez que les proporciona la apariencia de una capacidad de cálculo mucho mayor.

En el caso de infraestructuras reales, lo que ha sido referido como "usuarios" son en la práctica centros de investigación o universidades, y la cooperación de usuarios se lleva a cabo en el marco de las denominadas Organizaciones Virtuales (VO).

A continuación, todos estos conceptos serán definidos de una manera rigurosa, junto con una descripción exhaustiva de la tecnología necesaria para llevar a cabo la deseada compartición de recursos.

2.1.1. Qué es la computación en Grid

El problema que la Grid intenta resolver es "coordinar la compartición de recursos y resolución de problemas en organizaciones virtuales dinámicas y multi-institucionales" (1). Por supuesto, esta compartición tiene que estar altamente controlada, de modo que tanto los usuarios como los proveedores de servicios sepan claramente qué, cuánto, a quién y bajo qué condicio-

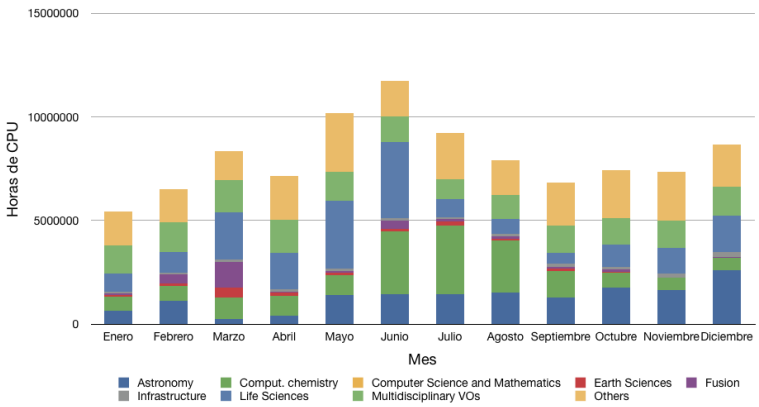

Figura 2.1: Demanda computacional en la infraestructura EGI a lo largo del 2010.

nes se comparte un determinado recurso. Estas condiciones definen grupos de usuarios denominados *organizaciones virtuales* -definidas de una manera mÃ¡s rigurosa en la sección 2.2.3.

Con esto en mente, una definición intuitiva de la Grid puede ser: mientras que la World Wide Web es un servicio para compartir información en Internet, la Grid es un servicio para compartir recursos de computación y capacidad de almacenamiento en Internet.

Una definición más clásica se puede encontrar en (2). Según ella, una infraestructura Grid sería un sistema que:

- Coordina recursos que no están sujetos a un control centralizado.
- Usa protocolos e interfaces estándar, abiertos y de uso general.
- Proporciona calidades de servicio no triviales.

El primer punto implica que los recursos no pertenecen a una misma persona u organización, por lo que no se tiene un control absoluto de los mismos. Por el contrario, el uso de dichos recursos está supeditado a la autorización del administrador de los mismos, así como el software instalado, política de seguridad o cualquier otro criterio.

El segundo punto deja claro que la Grid pretende ser un entorno de uso general, no orientado a un determinado tipo de aplicación o ámbito -tal como la compartición de ficheros, por ejemplo. Para ello es fundamental que sea un entorno abierto, de modo que cada cual pueda adaptarlo a sus necesidades, y pueda ser implementado en el mayor número de arquitecturas y sistemas posible. El hecho de que los protocolos sean estándar evita la fragmentación, es decir, que haya diferentes redes Grid utilizando distintos protocolos y no puedan colaborar.

Y el tercer punto implica que la Grid *funcione*. De nada sirve todo lo demás si no se es capaz de proveer una infraestructura estable, en la que sus usuarios puedan confiar y que proporcione una utilidad mayor que la de la suma de sus partes. Después de todo una Grid no es un fin en sí mismo, sino un medio que proporciona a sus usuarios los recursos informáticos que necesiten para llevar a cabo sus tareas.

La experiencia dice que es posible cumplir estos puntos obteniendo una infraestructura de calidad, que proporciona un servicio real a los usuarios y que sirva para llevar a cabo experimentos complejos. En este ámbito cabe destacar el proyecto "The Grid Workloads Archive" (3), que lleva 15 años monitorizando el estado de más de 150 *sites*, empleados por más de 2000 usuarios para ejecutar unos diez millones de tareas.

2.1.2. Qué NO es la computación en Grid

En la presente Tesis doctoral se utiliza una definición muy precisa de Infraestructura Grid. Por tanto, se ha considerado necesario el razonar por qué algunos tipos de infraestructura han quedado excluidos.

Primero, se consideran los sistemas de manejo de clúster tales como Sun Grid Engine (4) o Portable Batch System (5). Estos sistemas permiten el interconectar una serie de recursos distribuidos en un computador paralelo o en una red local, proporcionando una QoS (*Quality of Service*, Calidad de Servicio) no trivial. Sin embargo no cumplen un principio fundamental para ser considerados una infraestructura Grid: los recursos están sujetos a un control centralizado, formando parte de la misma organización. Sin embargo sí que pueden formar parte de una infraestructura Grid, siendo el clúster considerado un solo recurso.

Los sistemas de computación voluntaria tales como el SETI@HOME (6) o Ibercivis (7), aunque en ocasiones han sido considerados infraestructuras Grid, tampoco son englobados en las definición antes propuesta. En este caso no hay un control centralizado sobre los recursos -que son aportados por cada voluntario- pero sí sobre su empleo. En el caso de Ibercivis, se utiliza un servidor BOINC (8) que se encarga de todas las tareas de gestión: *scheduler*, subida/bajada de ficheros, comprobación de los resultados parciales y demás. El software instalado en los equipos de los voluntarios que ceden tiempo de computación se conecta a este servidor para recibir la tarea a ejecutar y devolver el resultado. Puede verse una descripción completa de esta arquitectura en (7).

2.1.3. Tecnologías de computación en Grid

Durante la evolución de la computación Grid han surgido distintas alternativas para crear estas infraestructuras. Estas alternativas difieren en la cantidad y tipo de los servicios prestados. Como veremos, algunas de las

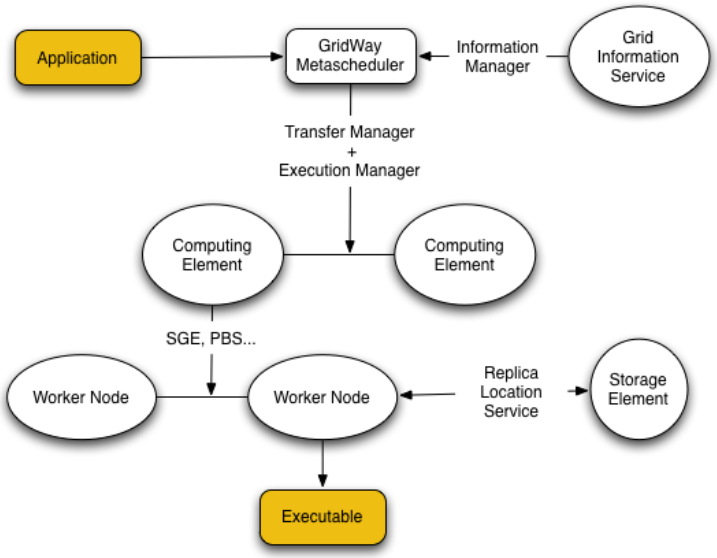

Figura 2.2: Arquitectura de una infraestructura Grid basada en GridWay.

soluciones propuestas únicamente sirven como un "esqueleto" sobre el que construir las aplicaciones, mientras que otras nacen con afán de totalidad y de proveer al usuario de todas las herramientas posibles para una ejecución sencilla de su trabajo.

A continuación se detallará la estructura típica de una infraestructura Grid, así como las alternativas más importantes -en el momento de la redacción de la presente Tesis Doctoral- a la hora de implementarla.

2.2. Arquitectura Grid

La figura 2.2 representa, de manera simplificada, la organización típica de una infraestructura Grid desde el punto de vista del usuario final. Esta figura integra todas las herramientas software que serán descritas a continuación y permite al lector visualizar la estructura de una manera gráfica.

En este caso, el centro de la infraestructura es el metaplanificador GridWay (9). Como se explicará más adelante, su trabajo consiste en decidir el mejor *site* donde ejecutar una determinada tarea y comprobar que todas las fases de esta ejecución son realizadas correctamente.

Para ello, la principal herramienta empleada es el GIS (*Grid Information Service*, Sistema de Información para Grid). GIS consiste en un servicio público que los recursos de la infraestructura emplean para anunciar su estado en cada momento. Así, un usuario puede decidir dónde ejecutar cada tarea

contando con la máxima información posible.

Cuando se ha decidido dónde enviar la tarea, se emplean las herramientas de Globus Toolkit (10) para autenticarse, copiar los ficheros necesarios al elemento remoto de cómputo (*Computing Element*), y llevar a cabo la ejecución en el nodo (*Worker Node*) que el planificador local decida.

En el caso de que la aplicación a ejecutar requiera un gran volumen de datos de entrada o genere muchos datos de salida, es posible utilizar elementos de almacenamiento (*Storage Elements*) donde disponer de esta información y poder manejarla de manera descentralizada. Esto permite minimizar el tráfico entre el recurso local y todos los remotos, eliminando así un posible cuello de botella y agilizando la ejecución todo lo posible.

Como veremos a continuación, estas operaciones pueden ser realizadas de diferentes maneras, dependiendo del middleware empleado. En el entorno de sistemas distribuidos, se puede definir middleware como "la capa de software que se encuentra entre el sistema operativo y las aplicaciones a ambos lados del sistema" (11).

2.2.1. Globus Toolkit

Globus Toolkit (10) es el estándar de facto en la computación Grid. Está compuesto por un conjunto de servicios, librerías y herramientas de desarrollo diseñadas para construir aplicaciones basadas en la Grid (12).

La mayor ventaja de Globus frente al resto de alternativas es la simplicidad: está constituido por una serie de comandos extremadamente simples que realizan pequeñas tareas de un modo eficiente. Esto ha permitido su empleo en muchos ámbitos, así como la creación de multitud de aplicaciones y middleware que extienden su funcionalidad basándose en estos comandos, intentando así proporcionar herramientas más poderosas y fáciles de manejar para el usuario final.

El diseño de la versión 4, GT4, se muestra en la figura 2.3 (10), y está detallado en los siguientes apartados.

2.2.1.1. Seguridad

El grupo de componentes de Globus Toolkit 4 denominado GSI (*Grid Security Infrastructure*, Infraestructura de Seguridad para Grid) se encarga de establecer medidas de seguridad en la infraestructura Grid. En concreto, sus responsabilidades son:

- *Comunicaciones.* Asegurar la integridad y seguridad de las comunicaciones entre *sites*.
- *Autenticación y Autorización.* Controlar quién puede acceder a cada recurso o servicio Grid; gestionar la existencia de VO, así como la política de seguridad de cada una de ellas.

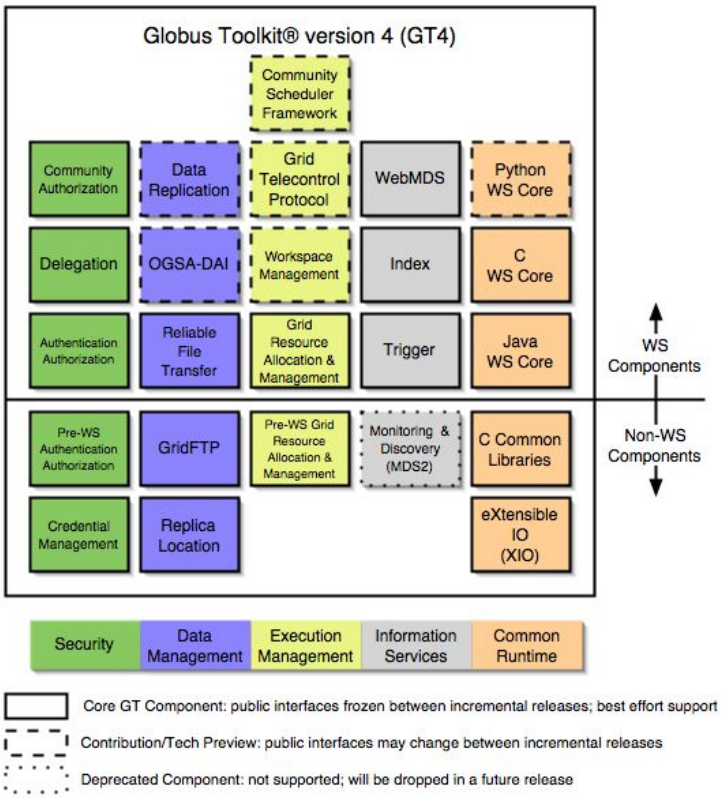

Figura 2.3: Arquitectura de Globus Toolkit 4.

- *Manejo y delegación de credenciales*. Crear credenciales de seguridad de los usuarios que no tengan acceso a una CA (*Certification Authority*, Autoridad de Certificación) externa; delegar las credenciales de seguridad del usuario de forma temporal.

2.2.1.2. Manejo de datos

Proporciona las herramientas necesarias para el manejo de datos dentro de una infraestructura Grid.

- *Grid File Transfer Protocol* (GridFTP) y *Reliable File Transfer Service* (RFT). GridFTP es un servicio FTP seguro; RFT es un servicio web de transferencia de ficheros que utiliza GridFTP para conseguir el mejor rendimiento posible, teniendo en cuenta las necesidades particulares de la Grid (grandes cantidades de datos en entornos dinámicos).

- *Réplica de datos*. Aasegurarse de que los datos están replicados en diferentes servidores para evitar su pérdida, permitiendo al usuario la

localización de los mismos de manera eficiente.

2.2.1.3. Control de la ejecución

Estos componentes de GT4 se encargan del lanzamiento, planificación y monitorización de los trabajos. Los más importantes son:

- GRAM (*Grid Resource Allocation and Management*, Administración y Localización de Recursos en Grid) . Proporciona servicios para lanzar y monitorizar trabajos en la Grid.
- CSF (*Community Scheduler Framework*, *Framework* de Planificacón para Comunidades). Proporciona una interfaz única para acceder a los gestores de carga locales, tales como *Portable Batch System* (PBS) o *Sun Grid Engine* (SGE).

2.2.1.4. Servicios de información

Los servicios de información de GT4 son un conjunto de herramientas para monitorizar y descubrir recursos dentro de una organización virtual. Están compuestos por un *Index Service*, utilizado para recuperar información de los distintos recursos de una VO y un *Trigger Service*, que además de recuperar los información de los recursos, puede desempeñar determinadas tareas basándose en esos datos.

2.2.2. gLite

gLite (13) es un middleware para computación Grid surgido en el seno del proyecto EGEE (14). De acuerdo con la información presente en la web corporativa del proyecto, gLite ha sido creado gracias a los esfuerzos colaborativos de más de ochenta personas de doce instituciones académicas y centros de investigación públicos y privados, con el propósito de constituir un *framework* para construir aplicaciones Grid que permitan aprovechar los recursos de computación y de almacenamiento distribuidos entre todas las entidades que colaboran en el proyecto.

gLite ha sido creado integrando diferentes fuentes. Además del código desarrollado dentro del proyecto utiliza componentes de otros muchos, incluyendo LCG (15) y VDT (16). gLite es un software distribuido, basado en la construcción de diferentes servicios que se hayan distribuidos a lo largo de la infraestructura. Gracias a esos componentes (*node types*, tipos de nodo) y al empleo de una única distribución de Linux, se pretende alcanzar una instalación y configuración sencillas de los servicios que se deseen. Actualmente esta plataforma es Scientific Linux 5, existiendo versiones no oficiales (las llamadas *ports*) para otras distribuciones tales como Debian, RedHat o SUSE (17).

gLite está distribuido bajo licencia Apache 2.0, lo que asegura tanto su distribución libre como la posibilidad de empleo para proyectos lucrativos. Con esto se pretende que pueda ser acogido por un amplio número de usuarios, tanto de la comunidad científica -muchas veces limitada al empleo de herramientas libres- como en el ámbito empresarial, donde el poder modificar el código y adaptarlo a proyectos propios sin la necesidad de liberarlo de nuevo puede ser necesario.

Actualmente gLite puede abarcar todas las fases necesarias para la ejecución de aplicaciones en Grid. Para ello integra componentes de varios proyectos de middleware como Globus Toolkit y CondorCondor-G (18), además de componentes desarrollados para el proyecto LCG. También incluye tanto herramientas de bajo nivel para asegurar la compatibilidad con *schedulers* como PBS, Condor y LSF, como servicios de alto nivel para facilitar la construcción de aplicaciones Grid.

La figura 2.4 (19) proporciona una visión detallada de la arquitectura de gLite, donde los diferentes módulos quedan patentes.

- El primero, UI (*User Interface*, Interfaz de Usuario), proporciona al usuario una interfaz basada en comandos de texto, que permite el envío de trabajos de forma sencilla.

- La inclusión del metaplanificador WMS *(Work Management Service*, Servicio de Administración de Tareas) (20) permite el emplear gLite con un alto nivel de abstracción, delegando al sistema la decisión sobre el lugar donde se ejecuta cada tarea. Este servicio será explicado con detalle en la sección de metaplanificadores.

- Los tres tipos de *Computing Elements* (LCG (15), gLite (13) y CREAM CE (21)) hacen posible la ejecución en Grid de los trabajos en distintas plataformas.

- Por último, el software instalado en los Worker Nodes es el encargado de llevar a cabo la ejecución de los trabajos, controlando su correcta ejecución e informando de los errores o problemas que hayan podido surgir.

Los servicios Grid de gLite están basados en SOA (*Service Oriented Architecure*, Arquitectura Basada en Servicios) (22), lo que facilita la conexión el software con otros servicios Grid y que será posible adoptar los estándares de Grid presentes y futuros, tales como el WSRF (*Web Service Resource Framework*, Entornos de Desarrollo de Servicios Web con Recursos) (23) de OASIS (*Organization for the Advancement of Structured Information Standards*, Organización para el Avance de Estándares en Información Estructurada) (**?**) y el OGSA *(Open Grid Service Architecture*, Arquitectura de

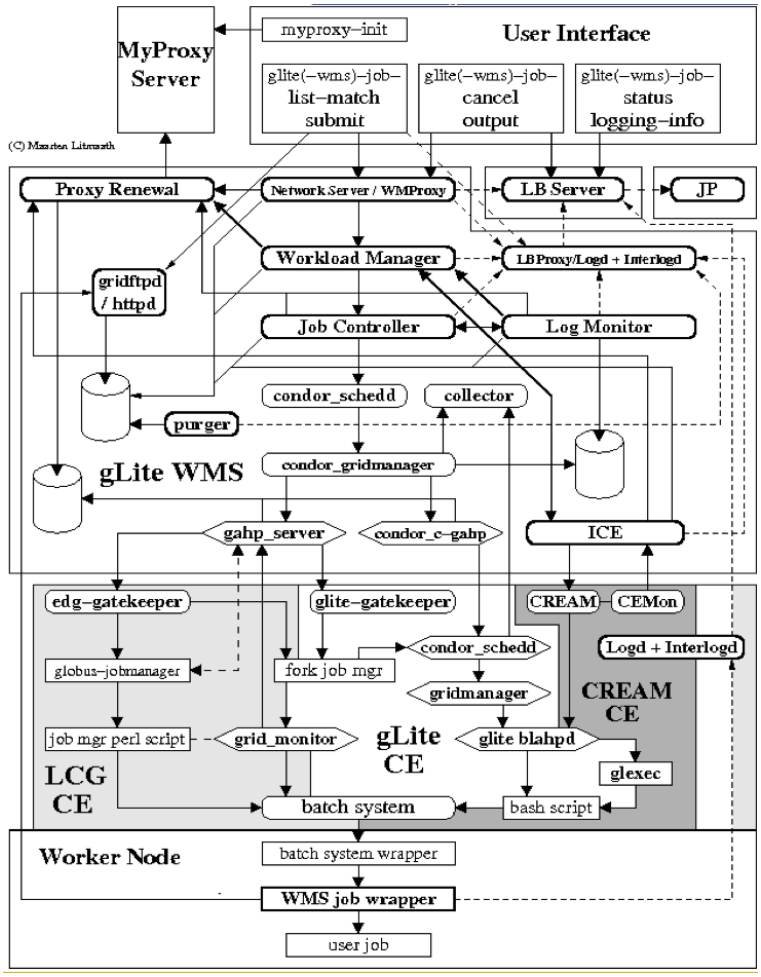

Figura 2.4: Arquitectura de gLite.

Servicios del Grid Abierto) (24) del Global Grid Forum (25). La pila de gLite ha sido diseñada como un sistema modular, permitiendo a los usuarios desplegar diferentes servicios de acuerdo a sus necesidades, en lugar de ser forzados a usar el sistema completo.

La integración de CREAM con WMS se realiza a través del módulo separado ICE (*Interface to CREAM Environment*, Interfaz para el Entorno CREAM). ICE recibe las peticiones de trabajos y otras tareas del componente WMS y utiliza los métodos apropiados de CREAM para llevar a cabo la operación solicitada. ICE es así mismo responsable de monitorizar el estado de las tareas enviadas y tomar las acciones apropiadas cuando se detecten cambios relevantes en el estatus de las tareas, por ejemplo, el reenviar una tarea si se ha detectado un fallo del nodo Grid donde se estaba ejecutando.

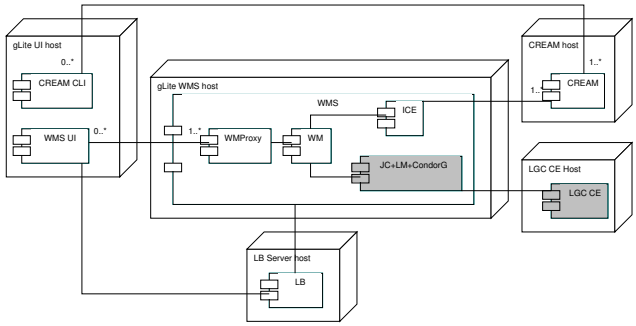

Figura 2.5: Envío de trabajos con gLite (simplificado).

La figura 2.5 (26) muestra un ejemplo de envío de trabajos con gLite, integrando CREAM con WMS. Aquí, los usuarios de gLite envían trabajos a través de la UI de gLite, que transfiere la petición al componente ICE local. ICE comprueba que el usuario posee un proxy VOMS (*Virtual Organization Membership Service*, Servicio de Membresía de la Organización Virtual) válido, esto es, una credencial que le identifica como usuario. A continuación, interactúa con CREAM empleando un servicio web. CREAM notifica al monitor CEMonitor sobre los cambios en el trabajo empleando su *backend* compartido. ICE se suscribe a CEMonitor para estar al tanto de las notificaciones, por lo que recibe todos los cambios de estatus a medida que ocurren y conoce el estado de los trabajos. El cambio en el estatus de los trabajos también es enviado al servicio *Logging and Bookeeping service* (LB), un mecanismo distribuido de seguimiento de trabajos.

Todos los componentes del middleware gLite (incluido CREAM y el monitor CEMonitor) son construidos utilizando el sistema ETICS (*e-Infrastructure for Testing, Integration and Configuration of Software*, e-Infraestructura para el Testeo, Integración y Configuración de Software) (27). ETICS es un sistema que permite automatizar la construcción, configuración, integración y testeo de software. Utilizando ETICS es posible integrar procedimientos, herramientas y recursos en una infraestructura coherente, proveyendo además de un acceso intuitivo a través de un portal web. éste permite a los desarrolladores ensamblar distintos componentes, creados de manera independiente, en un paquete de software. Cada uno de estos componentes puede usar su propio método para compilar y ensamblar el código (Make para C/C++, Ant para Java...), proporcionando ETICS un conjunto de comandos para manejar esto de manera homogénea.

2.2.2.1. CREAM

Los componente para el manejo de trabajos son empleados para enviar, cancelar y monitorizar trabajos que se ejecutan en un *Computing Element* adecuado. A continuación se describirá la arquitectura de CREAM (21) (*Computing Resource Execution and Management*, Administración y Ejecución de Recursos Computacionales), un sistema diseñado para manejar eficientemente un CE en un entorno Grid.

Para los usuarios es importante poder monitorizar sus trabajos. Esto significa conocer el estado de su ejecución su estado en cada momento, y en caso de haber cualquier problema poder obtener la mayor información posible acerca del mismo. CREAM proporciona herramientas para el envío y control de tareas en infraestructuras Grid, mientras que CEMonitor es un *framework* de propósito general para la notificación de eventos asíncronos. Ambos componentes proporcionan una interfaz mediante Servicios Web que permite a los clientes el envío, monitorización y control de tareas computacionales a un LRMS (*Local Resource Management System*, Sistema de Manejo de Recursos Locales).

Ambos componentes han sido integrados en WMS empleando ICE.

2.2.2.2. WMS

gLite WMS (*Workload Management System*, Sistema de Manejo de Cargas de Trabajo de gLite) (20) es la puerta de acceso de gLite para acceder a los recursos disponibles en una infraestructura Grid. Su meta es proveer un servicio fiable y eficiente para la distribución y gestión de trabajos en Grid, ocultando a los usuarios la complejidad intrínseca a este tipo de infraestructura. Ha sido diseñado con la intención de que sea genérico, pudiendo así dar soporte a aplicaciones de diferentes dominios.

Con el propósito de mantener homogénea la estructura de la presente Tesis Doctoral, una explicación detallada de WMS, incluyendo su arquitectura y funcionalidad, se puede encontrar en la sección perteneciente a Metaplanificadores.

2.2.3. Organizaciones Virtuales

En computación Grid, una VO (*Virtual Organization*, Organización Virtual) se refiere a una conjunto dinámico de individuos, usuarios y recursos que cooperan para forman una compartición de recursos flexible, segura y coordinada (1).

Como se ha descrito en la sección 2.1.1 “Qué es la computación Grid”, el problema que se intenta resolver es coordinar la compartición de recursos y resolución de problemas en organizaciones virtuales dinámicas y multi-institucionales. En esta compartición los proveedores de recursos han de de-

finir claramente qué recursos existen, para qué se pueden emplear y quién puede hacerlo bajo determinadas condiciones. Por su lado, los consumidores tienen que indicar qué recurso desean utilizar, en qué momento y para qué.

Este conjunto de normas, junto con los proveedores de recursos y los consumidores -sean individuos o instituciones- que deciden acatarlas, determinan una Organización Virtual. Así, estas organizaciones pueden tener ámbitos de actuación y composiciones muy distintas, dependiendo de los usuarios que la formen y el fin al que estén destinadas. Y por supuesto, tanto los usuarios como los proveedores de recursos pueden formar parte de más de una Organización Virtual, siempre que sus intereses sean concurrentes con los objetivos de estas.

2.2.3.1. EGI

El proyecto EGI (*European Grid Iniciative*, Iniciativa Grid Europea) (28), heredero de EGEE (*Enabling Grids for E-sciencE*, Habilitando Grids para la E-Ciencia) (14), reúne a científicos e ingenieros de más de 240 instituciones en 45 países alrededor del mundo para construir una infraestructura Grid con fines científicos que esté disponible permanentemente.

Sus objetivos se centran en tres áreas clave:

- Construir una infraestructura Grid consistente, robusta y segura.
- Mantener y mejorar el *middleware*, de forma que se proporcione a los usuarios la mejor experiencia de uso posible.
- Atraer nuevos usuarios tanto de la industria como del entorno científico, y asegurarse de que reciban toda la educación y soporte técnico necesarios que necesiten para emplear los recursos eficientemente.

En el momento en que la presente Tesis Doctoral fue redactada, EGI contaba con una impresionante cantidad de recursos (29): 330 *sites* que aportaban un total de 68.000 CPU, más de 15 PB de almacenamiento y una tasa de transferencia de más de 1.5 GB/s, que permitía la ejecución de 150.000 trabajos al día.

En ese momento existían en EGI 215 Organizaciones Virtuales con diferentes miembros, reglas a la hora de enviar trabajos y recursos accesibles. De todas ellas, se utilizaron los recursos de dos a la hora de elaborar este trabajo: *dteam* (30) durante el desarrollo del software (es una cola donde se permiten las pruebas y trabajos que no sean de producción) y *fusion* (31) una vez que la aplicación estaba en producción, con muchos más recursos que *dteam* y una infraestructura más estable.

Por su parte, en el momento de redactar esta memoria, la VO *fusion* disponía de 45 nodos con más de 15.000 CPU disponibles, de las que un tercio pertenecían al GridPP, UK Computing for Particle Physics.

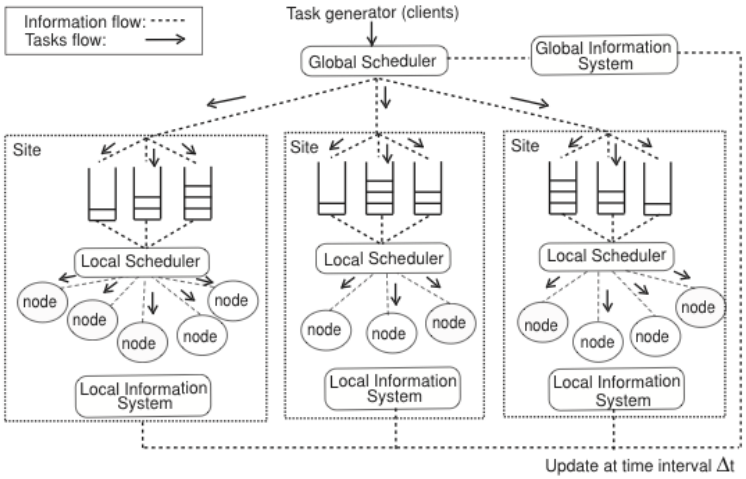

Figura 2.6: Ejemplo de arquitectura con un *scheduling* centralizado.

2.3. Metaplanificadores en Grid

La metaplanificación es una técnica software de computación para optimizar la carga computacional que combina múltiples DRMs (*Distributed Resource Managers*, Gestores de Recursos Distribuidos), en una sola vista agregada, permitiendo a los trabajos Batch el ejecutarse en el mejor recurso disponible.

De una manera más sencilla, puede verse de la siguiente manera; mientras que la planificación se encarga de distribuir tareas entre diferentes nodos de un clúster, la metaplanificación se encarga de distribuir tareas entre diferentes planificadores.

Habitualmente, estos recursos locales emplean diferentes LRMs (*Local Resource Managers*, Gestores de Recursos Locales) tales como PBS (5), SGE (4) o Condor (18), y están distribuidos en una sola organización o en varias, en este caso formando una Organización Virtual. Así pues, los metaplanificadores modernos son capaces de comunicarse con diferentes LRMs, pudiendo así llevar a cabo una metaplanificación más eficiente y abarcando el mayor número de recursos posible.

La figura 2.6 (32) muestra la arquitectura básica de un sistema distribuido que incorpora un metaplanificador. Como se ve, está compuesto de un conjunto de *sites* más un planificador global (lo que aquí se define como metaplanificador). Este metaplanificador envía los trabajos a los distintos *sites*, que tienen varias colas dependiendo del tipo y origen del trabajo. Ahí, un planificador local se encarga de decidir qué tarea envía en cada momento a cada nodo. Por último, el LIS (*Local Information System*, Sistema de Información Local) informa al GIS (*Global Information System*, Sistema de

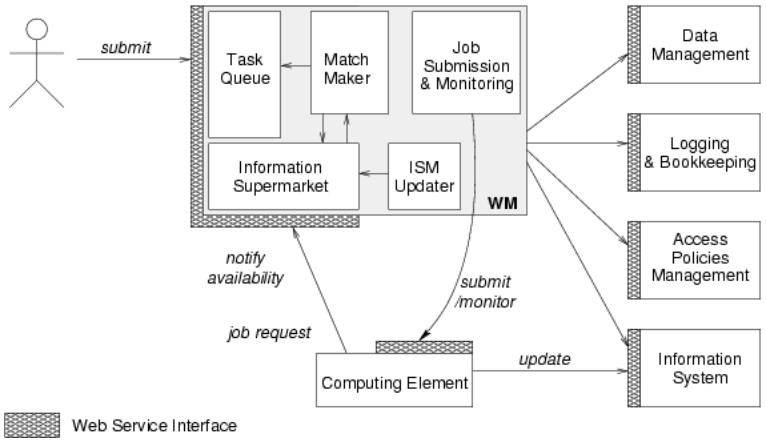

Figura 2.7: Arquitectura de WMS.

Información Global) sobre el estado de cada *site*, repitiendo esta operación cada cierto periodo de tiempo. El metaplanificador emplea esta información para tomar las decisiones de la manera más precisa posible.

En este punto cabe destacar que, en el caso de algunos metaplanificadores como GridWay, es posible implementar esta arquitectura de manera recursiva, donde un metaplanificador envía trabajos a otro, como si de un planificador local se tratara. En cualquier caso el estudio de este tipo de propuestas se escapa del ámbito de la presente Tesis Doctoral. Para un estudio en profundidad del tema, puede encontrarse más información en el trabajo de Huedo (33) y las referencias que ahí se proponen.

2.3.1. WMS

WMS (*Workload Management System*, Sistema de Manejo de Cargas de Trabajo) (34) engloba un conjunto de componentes de middleware Grid responsables de la distribución y gestión de tareas en infraestructuras Grid, intentando que su ejecución sea lo más eficiente y rápida posible.

La figura 2.7 (34) muestra la arquitectura de WMS.

Como se puede apreciar, el componente central es el WM (*Workload Manager*, Gestor de Carga de Trabajo), cuyo propósito es recibir las peticiones de ejecución del usuario, decidir el mejor *site* donde llevar esta ejecución a cabo y controlar todas las etapas de esta ejecución.

Para ello, la *Task Queue* (Cola de Tareas) recibe las peticiones de los usuarios. El módulo *Match Maker* (Verificador de Concordancias) determina, basándose en la información que le proporciona el *Information Supermarket* (Supermercado de Información) el lugar y momento propicios para ejecutar esta tarea y cuando está decidido, el módulo de *Job Submission &*

Monitoriong (Envío y Monitorización de Trabajos) se encarga de controlar esta ejecución.

El resto de los componentes que aparecen en la figura 2.7 se encargan de tareas auxiliares necesarias para la ejecución. Así, hay módulos para el manejo de datos, el registro de operaciones, el control de acceso y el sistema de información.

WMS ha sido diseñado y desarrollado con la intención de ser el punto de entrada principal a los recursos Grid disponibles en una infraestructura basada en gLite. El objetivo es proveer un entorno robusto y eficiente para la distribución y ejecución de trabajos en Grid, ocultando al usuario final la complejidad intrínseca a este tipo de infraestructuras. La abstracción que proporciona es suficiente para soportar aplicaciones de dominios muy diferentes.

2.3.2. GridWay

En el apartado 2.2.1 se ha descrito Globus Toolkit, mostrando cómo proporciona un conjunto de herramientas que permiten al usuario utilizar una infraestructura Grid. Sin embargo, este usuario es responsable de efectuar manualmente todas las operaciones necesarias para poder ejecutar un trabajo: descubrir los recursos disponibles, decidir dónde se ejecutará el trabajo, enviarlo, recoger los datos de salida, etc. Además Globus Toolkit no proporciona soporte para la migración de trabajos ni la recuperación de fallos, un aspecto clave a la hora de trabajar eficientemente.

Las infraestructuras Grid son, por definición, dinámicas. Por tanto, las aplicaciones que dependen de ellas han de ser capaces de adaptarse a un entorno cambiante, aprovechando los nuevos recursos que aparecen y recuperándose de fallos de los que esté empleando en un momento dado. Para ello, en la literatura (9) se han propuesto dos técnicas fundamentales:

- Planificación adaptable que envíe trabajos a los recursos Grid teniendo en cuenta sus recursos físicos, disponibilidad y funcionamiento durante los trabajos previamente enviados a ese recurso.

- Ejecución adaptable que pueda migrar trabajos entre diferentes recursos teniendo en cuenta tanto eventos internos de la aplicación (necesidades cambiantes) como de la Grid (aparición de nuevos recursos o desaparición de los existentes).

GridWay (35) es una meta-planificador que permite una ejecución sencilla de trabajos en un entorno Grid dinámico, llevando a cabo todos los pasos necesarios. El objetivo es que el usuario final pueda definir sus necesidades de una manera sencilla, delegando en Gridway la responsabilidad de que el trabajo sea ejecutado eficientemente. De este modo la utilización de

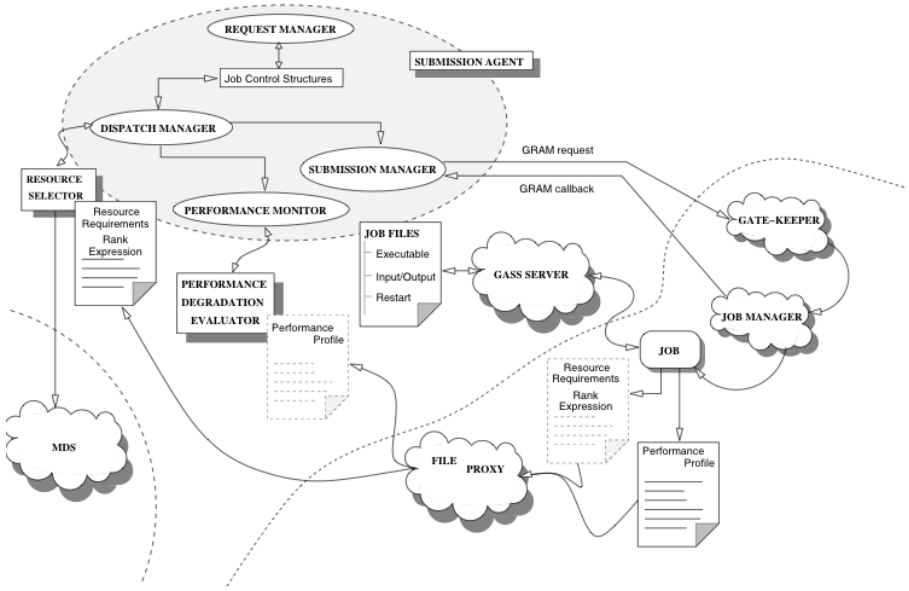

Figura 2.8: Arquitectura de GridWay.

una infraestructura Grid deja de estar restringida a usuarios expertos y un abanico mucho más amplio de personas puede utilizarla para satisfacer sus necesidades de computación.

Para lograr este objetivo, el núcleo de GridWay es un agente que realiza todas las labores relacionadas con la ejecución remota de trabajos y supervisa que la ejecución del trabajo sea correcta. La adaptación a condiciones cambiantes se realiza mediante una planificación y ejecución adaptables. En concreto, una vez que un trabajo ha sido enviado a un recurso en concreto puede ser migrado por las siguientes razones:

- Relacionadas con la Grid: un recurso mejor es descubierto, el recurso original falla o el trabajo es cancelado o suspendido por el administrador del recurso.
- Relacionadas con la aplicación: se produce una disminución del rendimiento, cayendo por debajo de unos límites establecidos, o los requisitos de la aplicación cambian.

Un importante objetivo de este proyecto es crear un *framework* que permita a usuarios no expertos ejecutar trabajos en Grid. En este contexto, GridWay supone una importante herramienta, ya que permite que el usuario se centre en el problema a resolver y no en la tecnología subyacente.

En la figura 2.8 (36) se puede observar la arquitectura de GridWay, con los principales módulos que componen la aplicación.

Su funcionamiento es, en esencia, el mismo que el de WMS. Primero, el *Request Manager* (Gestor de Peticiones) recibe del usuario el trabajo a ejecutar; el *Resource Selector* (Seleccionador de Recursos) determina el mejor recurso para dicho trabajo y lo coloca en una cola de envíos; el *Dispatch Manager* (Gestor de Despacho) periódicamente intenta despachar todos los trabajos que el Resource Manager dejó encolados; el *Submission Manager* (Gestor de Envíos) lleva a cabo el envío, en caso de ser posible; por último, el *Performance Monitor* (Monitor de Rendimiento) comprueba periódicamente la salud de los recursos remotos y el estado de los trabajos enviados, con el fin de informar al *Submission Manager* de cualquier posible problema en su ejecución. El resto de los módulos y ficheros que aparecen en la figura 2.8 realizan tareas auxiliares y de apoyo al *Submission Agent* (Agente de Envíos).

Mediante el empleo de GridWay sobre Globus en lugar de WMS sobre gLite el aumento de rendimiento es notable. Existe abundante bibliografía en el área (37) (38) que lo demuestra.

Una funcionalidad destacable de GridWay es la integración con DRMAA (39) *(Distributed Resource Management Application API*, API para Aplicaciones de Manejo de Recursos Distribuidos), que se realiza de un modo sencillo y eficiente. La implementación de esta interfaz en librerías para algunos de los lenguajes más extendidos (Java, C/C++, Python, Ruby y Perl), permite que cualquier aplicación que la utilice pueda enviar y controlar trabajos con GridWay de manera extremadamente sencilla: basta con indicar la localización de la librería deseada en el momento de compilar la aplicación.

2.4. Paradigmas de desarrollo de aplicaciones en Grid

Lo que los usuarios esperan de la Grid es un acceso sencillo, facilidades en la creación y envío de trabajos y alto rendimiento (40).

El primer punto implica protocolos ágiles y rápidos para el registro, autenticación y autorización. Puede ser llevado a cabo gracias a herramientas tales como interfaces de usuario sencillas (41). Actualmente, este área está cubierta gracias a interfaces de línea de comandos presentes tanto en en Globus Toolkit como en gLite: con un único comando y una clave de usuario es posible llevar a cabo estos pasos de una manera automática.

La creación, envío y monitorización de trabajos deben poder ser realizados empleando interfaces sencillas. En concreto, los desarrolladores han de disponer de un conjunto mínimo de API o interfaces gráficas para construir sus aplicaciones (42). El resultado de cara al usuario final ha de ser una aplicación amigable y eficiente a la hora de ejecutarse en Grid.

Para llevar a cabo esta tarea existen diversas herramientas, algunas de las cuales se detallan en el resto de este apartado. Aunque todas tienen par-

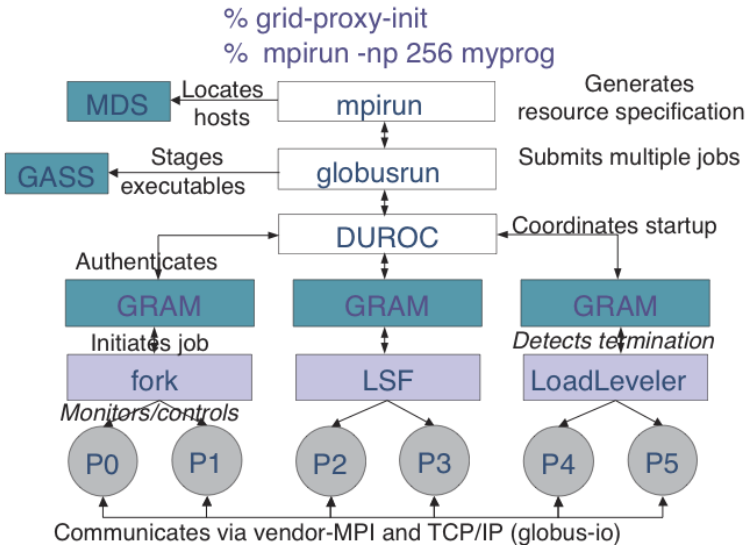

Figura 2.9: Arquitectura de MPICH-G2.

ticularidades, parten de una idea común: permitir al desarrollador abstraerse de los detalles de la ejecución en Grid, proporcionándole una interfaz de programación estándar de alto nivel. De esta manera, las tareas de creación y mantenimiento de aplicaciones se ven enormemente simplificadas al encapsular todas las operaciones relacionadas con la Grid aislándolas del código de la aplicación.

Mediante el uso de estas herramientas, el desarrollador es capaz de enviar fragmentos de la aplicación -en forma de aplicaciones independientes o *scripts*- a *sites* remotos, recoger los resultados de su ejecución, y proseguir con la ejecución local en caso de que fuera necesario. Así, esta aproximación permite liberar al usuario final de todas las tareas relacionas con Grid, proporcionándole una interfaz potente y de manejo sencillo.

2.4.1. MPICH-G2

MPICH-G2 (43) es una implementación del estándar MPI v1.1 (44) adaptada a su ejecución en Grid. Esto es, utilizando servicios de Globus Toolkit, MPICH-G2 permite al desarrollador crear aplicaciones que se ejecuten paralelamente en diferentes *sites*, corriendo aplicaciones MPI. MPICH-G2 emplea MPI propietarios para las llamadas MPI entre tareas dentro del mismo clúster, y convierte los mensajes entre tareas de diferentes clústers en mensajes que envía mediante protocolo TCP.

Para los mensajes entre *sites*, MPICH-G2 emplea los servicios que provee Globus Toolkit 2.X.

La figura 2.9 (43) muestra el funcionamiento de MPICH-G2 una vez que comienza la ejecución de una aplicación. El comando `mpirun` comienza la ejecución de la aplicación, y emplea `globusrun` para copiar los archivos necesarios a los diferentes *sites* y ejecutar los trabajos. La librería DUROC emplea GRAM para arrancar y controlar las tareas deseadas. Por último, tras pasar por el gestor local de tareas, éstas se ejecutan en el *slot* que les corresponda. Como se puede ver, la comunicación entre las diferentes tareas MPI se lleva a cabo utilizando dos protocolos distintos, según la posición relativa de las tareas a comunicar.

El principal problema que presenta MPICH-G2 es el rendimiento y la continuidad del proyecto. Al ejecutar aplicaciones MPI estándar, si éstas no están diseñadas teniendo en cuenta las particularidades de las infraestructuras Grid (recursos heterogéneos, propensos a fallos y extremadamente dinámicos) la alta latencia y diferencias de rendimiento entre los diferentes *sites* hacen que el rendimiento de la aplicación sea muy bajo, con lo que se pierden la principal ventaja de la ejecución en Grid.

Por otro lado, al menos aparentemente el proyecto no presenta una continuidad en su desarrollo y mantenimiento. En el momento de la redacción de esta memoria (junio del 2011) su página web (45) llevaba 5 años sin actualizarse, los enlaces a la documentación eran incorrectos, y aparecían referencias a trabajos futuros que 4 años después del *deadline* previsto seguían sin ser concluidos.

2.4.2. SAGA

SAGA (*Simple API for Grid Applications*, API Simple para Aplicaciones Grid) (46) es un estándar abierto definido y mantenido por el Open Grid Forum. Su función es describir una interfaz de alto nivel para la creación de aplicaciones Grid.

El propósito de SAGA no es reemplazar Globus o cualquier otro middleware y su público objetivo no son los desarrolladores de middleware. Al contrario, pretende actuar como una capa adicional que proporcione un mayor nivel de abstracción, ayudando a desarrollar códigos que funcionen en Grid sin necesidad de aprender nuevas técnicas o herramientas.

SAGA está estandarizado por el OGF. Se compone de una serie de documentos OGF:

- *Casos de Uso*: un documento que describe los casos de uso objetivos de SAGA.
- *Análisis de Requimientos de SAGA*: un documento que extrae requerimientos específicos a partir del anterior, Casos de Uso.
- *Núcleo del API de SAGA*: las bases del estándar.

- *Extensiones al API de SAGA*: una serie de extensiones para SAGA que mantienen su filosofía y añaden funciones auxiliares.

En el momento de la redacción de esta memoria, existían bindings de SAGA para Java, C/C++ y Python.

El principal problema de SAGA, por lo que fue descartado a la hora de realizar este trabajo, es que obvia todas las facilidades y funcionalidades que proveen los metaplanificadores detallados en la sección anterior.

2.4.3. GridRPC

En 2002 el Global Grid Forum propuso, a través de su Grupo de Investigación en Modelos de Programación, el API GridRPC (47) (*Grid Remote Procedure Call*, Llamadas a Procedimientos Remotos en Grid). Este API pretende estandarizar e implementar un sistema portable y simple de RPC. De este modo permite el desarrollo de aplicaciones en Grid a través de la comunicación entre el usuario y el middleware, abstrayendo al programador de la comunicación entre los elementos dinámicos de la Grid.

El API de GridRPC está dividida en dos partes con distintos objetivos según sus funcionalidades. Una parte se encarga de proporcionar herramientas a los usuarios finales, potentes y lo más simples posibles. Otra está orientada a los desarrolladores de middleware, más compleja pero flexible y eficiente, de manera que puedan emplearlas para construir nuevas herramientas.

GridRPC ha sido implementado por proyectos que pretenden incrementar el dinamismo de las aplicaciones que lo emplean, tales como NetSolve (48) y Ninf-G (49). Estas implementaciones ofrecen al programador infraestructuras para el desarrollo y despliegue de funciones y procedimientos remotos, facilitando al desarrollador la gestión de dichas funciones y procedimientos en los nodos remotos.

Sin embargo, al igual que pasa con SAGA, Grid-RPC no consigue abstraer al desarrollador de las tareas relacionadas con la gestión de recursos., Esto hizo que fuera desechado para la realización del presente trabajo.

Es importante destacar que Grid-RPC constituye, junto con DRMAA, el primer documento en alcanzar el status de "Recomendación para Grid" del Open Grid Forum (50). Sin embargo, por la naturaleza de la presente investigación, DRMAA se adapta mejor a las necesidades de la aplicación que se desea desarrollar.

2.4.4. DRMAA

DRMAA *(Distributed Resource Management Application API*, API para Aplicaciones de Manejo de Recursos Distribuidos) (39), es un API empleado para especificar el envío y control de tareas a uno o varios recursos en una infraestructura de memoria distribuida. Permite al programador el abstraerse

de las tareas de manejo de trabajos, definiendo operaciones que cubren todos los pasos de una ejecución en Grid: envío, monitorización y control de los trabajos.

DRMAA permite crear aplicaciones que emplean recursos distribuidos de una manera sencilla y práctica. Gracias a un API de alto nivel y su implementación por multitud de planificadores y metaplanificadores, los códigos creados con este API son fácilmente portables a distintas plataformas computacionales distribuidas.

Una de las ventajas de DRMAA es que permite desacoplar la aplicación del *scheduler* e infraestructura donde será ejecutada. Sirva como ejemplo el trabajo de Loureiro *et al.* (51) donde un mismo código ha sido ejecutado de manera eficiente en clúster y Grid con un trabajo de portado mínimo. Dado que en la actualidad las infraestructuras evolucionan y se desarrollan de una manera sorprendentemente rápida, el empleo de DRMAA permite asegurar con cierto grado de certeza que el código creado podrá ser utilizado en el futuro y no quedarse desfasado en un corto plazo de tiempo.

Por otro lado, el empleo de un API para separar la aplicación en sí de la parte de gestión de recursos -planificación, copia de ficheros, etc.- simplifica enormemente la tarea del desarrollador de la aplicación. Y el delegar estas operaciones en un metaplanificador externo hace que se puedan aplicar técnicas de planificación eficientes, empleando más información de la que usaría el usuario medio, y eligiendo el algoritmo más adecuado según el tipo de problema a resolver.

Por último, el poder crear aplicaciones locales que empleen toda la potencia del Grid de manera transparente al usuario final hace que el resultado sea, además de eficiente, usable y de fácil manejo.

2.4.4.1. Especificación del estándar

El estándar DRMAA abarca una serie de operaciones relacionadas con la ejecución de trabajos en entornos distribuidos.

La especificación DRMAA fue diseñada de manera que suponga una interfaz ligera y modular para sistemas clúster y Grid. Así, DRMAA se creó intentando mantener el API lo más simple posible, manteniendo la interoperabilidad entre los sistemas DRM más usados aunque limitando al máximo el número de funciones y conceptos incluidos. Así mismo la especificación se realiza con una sintaxis abstracta basada en funciones procedurales, de manera que pueda ser implementada con el lenguaje de programación que se desee.

Las funciones de DRMAA pueden agruparse según su utilidad de la siguiente manera:

- *Envío y monitorización de trabajos.* Estas funciones se encargan de la gestión de trabajos. Pueden enviarse uno o más trabajos con una sola

función -y en este caso, pueden ser idénticos o paramétricos- así como consultar el estado de los trabajos enviados. Dentro de este grupo se incluyen funciones como `drma_run_job()` o `drma_job_ps()`.

- *Control de trabajos.* Este grupo de rutinas controla la ejecución de trabajos y su correcta ejecución. Para ello incluye funciones que permiten manipular el trabajo -pausarlo, reiniciar su ejecución y forzar su terminación, por ejemplo. Otro grupo de funciones fuerza que la aplicación espere a que termine un trabajo o un grupo de ellos, permitiendo así la creación de barreras y la verificación de la correcta ejecución. En este caso `drma_control()` es la función que permite controlar la ejecución y `drma_synchronize()` permite establecer barreras. En ambos casos su comportamiento puede ser modelado gracias a un amplio abanico de parámetros.

- *Rutinas Auxiliares.* El último grupo de rutinas se encarga de tareas auxiliares que, sin aportar nuevas funcionalidades al estándar, facilitan la creación de aplicaciones que utilicen DRMAA. Entre ellas se incluyen funciones para conocer la versión de DRMAA soportada, el nombre del DRMS utilizado y rutinas para establecer contacto entre la aplicación con DRMAA y el DRMS. Así, este grupo incluye `drmaa_version()`, `drmaa_get_DRM_system()` y otras.

2.5. Modelo de programación DRMAA-GridWay

Gracias a que GridWay implementa el API de DRMAA en los lenguajes de programación más extendidos (Java, C/C++, Python y Ruby), el empleo de este metaplanificador en aplicaciones con DRMAA es inmediato: basta con señalar la localización de la librería de GridWay en tiempo de compilación. A partir de ese momento, GridWay es el encargado de llevar a cabo todas las operaciones relacionadas con la ejecución eficiente de los trabajos en Grid.

Dado que el API DRMAA es de muy alto nivel, es posible delegar en el metaplanificador gran parte de las decisiones a tomar. Esto supone una clara ventaja frente al resto de las alternativas, y asegura que la ejecución de los trabajos en Grid se realizará de una manera eficiente. El metaplanificador elige dónde y cuándo se ejecutará cada trabajo, y dado que conoce el estado de la infraestructura en cada momento y las particularidades del trabajo, la toma de decisiones es al menos tan eficiente como si la hubiera desarrollado el usuario final. De todas formas, es posible restringir parte de esta libertad -obligando a que un determinado trabajo se ejecute en un determinado *site*, por ejemplo- y que así un usuario avanzado puede hacer uso de esta funcionalidad en caso de que los trabajos a ejecutar posean particularidades que así lo requieran.

Por último, es destacable que DRMAA -al contrario que el resto de las alternativas aquí comentadas- sí hace uso de las herramientas que los metaplanificadores modernos ofrecen al usuario: es posible solicitar reenvío de tareas, control de errores o migración de trabajos en caso de pérdida de rendimiento de una manera sencilla. Este tipo de operaciones hace que el trabajo del desarrollador sea más cómodo y a la vez aumente la eficiencia del código, produciendo un resultado de mejor calidad de cara al usuario final.

Capítulo 3

Códigos de Monte Carlo

En el presente capítulo se describirá en profundidad el método de Monte Carlo. Comenzando con una introducción al método de Monte Carlo y su importancia en la computación actual, posteriormente se analizará su implementación en los llamados "códigos de Monte Carlo". Por último se estudiarán las posibilidades de implementar este tipo de códigos de manera que se puedan ejecutar de manera eficiente en infraestructuras Grid.

3.1. Introducción

El método de Monte Carlo adquiere su nombre del principado de Mónaco, "la capital del juego de azar", al tomar en sus inicios una ruleta como un generador simple de números aleatorios. El nombre y el desarrollo sistemático de los métodos de Monte Carlo data aproximadamente de 1946, con el desarrollo de la computadora electrónica. Sin embargo hay varias instancias (aisladas y no desarrolladas) en 1944 y en ocasiones anteriores.

El método de Monte Carlo surge con la creencia de que el comportamiento de fenómenos complejos puede ser modelado mediante la ejecución de múltiples simulaciones simples, empleado números aleatorios o pseudo-aleatorios para computar los resultados.

Gracias a esta particularidad el método de Monte Carlo es ampliamente empleado en diferentes áreas del conocimiento, resolviendo problemas donde el obtener un resultado exacto con un algoritmo determinista es imposible o inabarcable por su complejidad. Además, su modularidad simplifica su ejecución en entornos distribuidos, donde la distribución de tareas y su sincronización no es trivial. Como ejemplos de su uso en diferentes campos científicos se pueden nombrar casos de uso en medicina radiológica (52; 53), economía (54; 55), investigación medioambiental (56), física de altas energías (57; 58) o ingeniería espacial (59), entre otros.

En general, no existe un único método de Monte Carlo. Al contrario, el término describe un amplio conjunto de técnicas y aproximaciones. En

cualquier caso, todas ellas tienden a seguir un patrón característico:

- Definición del dominio de cada una de las entradas.
- Generación de entradas aleatorias dentro del dominio anterior y ejecución de una computación determinística con ellas.
- Cálculo del resultado final a partir de los resultados de las computaciones anteriores.

Como se verá, esta aproximación permite simplificar enormemente las simulaciones y su tiempo de ejecución, y al mismo tiempo obtener resultados con la misma precisión.

3.1.1. Evolución del método de Monte Carlo

3.1.1.1. Creación y primeros pasos

En 1945, en plena Guerra Mundial, tuvieron lugar dos acontecimientos que supusieron un profundo impacto en la ciencia y la tecnología: la primera detonación nuclear exitosa en el centro de Los Alamos (60) y la construcción del primer computador electrónico, el ENIAC (*Electronic Numerical Integrator And Computer*, Computador e Integrador Numérico Electrónico) (61). La combinación de ambos supuso un gran avance y sentó las bases de la computación actual.

La detonación de la primera bomba nuclear demostró que era posible crear una reacción de fisión capaz de autoalimentarse, produciendo una enorme cantidad de energía. Sin embargo, los fenómenos físicos detrás de este hecho aun no habían sido comprendidos y modelados completamente y, debido a la cantidad de recursos necesarios y la peligrosidad inherente a cada prueba, surgió la necesidad de poseer un modelo válido que permitiera avanzar lo más posible en su estudio en el plano teórico.

En aquel momento el ENIAC acababa de ser construido en la universidad de Pennsylvania, en un proyecto dirigido por el físico John Mauchly y el ingeniero Presper Eckert. Con sus 27 toneladas de peso y 167 m^2 de superficie, albergaba más de 17.000 válvulas de vacío, con las que era capaz de realizar cerca de 5.000 sumas y 300 multiplicaciones por segundo. En una época en la que las operaciones eran realizadas por enormes grupos de mujeres con calculadoras mecánicas de mesa, reglas de logaritmos y demás herramientas primarias, esto suponía un paso de gigante que abría las puertas a un nuevo conjunto de problemas que hasta el momento había resultado inabarcable.

John Von Neumann, profesor de matemáticas en el Institute for Advanced Study (62) estaba trabajando como consultor en Los Álamos. Ahí, mostró un enorme interés por los problemas termonucleares que su compañero y amigo Edward Teller y su grupo de investigación estaban intentando solucionar. Y

al enterarse de la existencia del ENIAC, consiguió convencer a las autoridades de que se lo cedieran para realizar los cálculos que sus investigaciones demandaban.

Originariamente el ENIAC había sido diseñado para el cálculo de trayectorias balísticas -lo que suponía una herramienta enormemente práctica durante la guerra- pero era capaz de realizar un abanico mucho mayor de operaciones. Von Neumann propuso emplear sus algoritmos para demostrar la utilidad real del equipo y en 1947 fueron finalmente ejecutados 9 experimentos complejos, simulando la colisión de neutrones bajo diversas circunstancias.

Para llevar a cabo estas simulaciones se empleó el Método de Monte Carlo, creado poco tiempo antes por Von Neumann y el doctor Stanislaw Ulam (63). Con el tiempo este método se convirtió en una herramienta fundamental para la ciencia y constituye uno de los pilares de la presente tesis doctoral.

La idea original del método de Monte Carlo surgió de una casualidad, la enfermedad de Ulam durante 1946 (63). Durante su convalecencia empleó el juego del solitario como manera de mantenerse distraído y rápidamente se dedicó a estudiarlo desde un punto de vista matemático. Ahí descubrió que le resultaba mucho más sencillo tener una idea general del resultado realizando múltiples pruebas y calculando las proporciones que los resultados que computando todas las posibles combinaciones y analizarlas formalmente. El siguiente paso fue aplicar esa misma técnica para su trabajo en Los Álamos sobre la difusión de neutrones, donde la solución exacta de todas las ecuaciones integro-diferenciales que guían el proceso es prácticamente imposible.

Aunque inicialmente Von Neumann fue reticente a emplear este nuevo método, pronto se convenció de sus virtudes y juntos se dedicaron a perfeccionarlo y estudiar las posibilidades que abría en distintas áreas. El nombre de "Monte Carlo" para fue propuesto por Nicholas Metropolis, y surgió como una broma interna (puede consultarse la historia completa en (64)) que hace referencia a la influencia del azar en la obtención de los resultados.

Desde ese momento y hasta el día de hoy el uso de los códigos de Monte Carlo -esto es, aplicaciones que emplean el método de Monte Carlo- no ha dejado de crecer. En (65) se indica que en 1987 se producían 10^{10} números aleatorios por segundo para la ejecución de códigos de Monte Carlo. Con una simple extrapolación y aplicando la ley de Moore, suponiendo que el porcentaje de cálculo dedicado a los MC es constante, supondrían 10^{25} números aleatorios por segundo a día de hoy. Calcular la veracidad de esta cifra está fuera del alcance y objetivos de la presente tesis doctoral, pero sirve para confirmar la plena vigencia del método en la computación actual.

3.1.2. Tendencias actuales de la computación de alto rendimiento

El objetivo final de esta tesis es desarrollar una aplicación y herramientas de análisis que trasciendan de un plano puramente académico y tengan una utilidad real. Para ello un punto fundamental es conocer el hardware sobre el que se ejecutará la aplicación y su previsible evolución, de manera que las innovaciones aquí planteadas sigan siendo válidas en el futuro.

Como es bien sabido, la potencia de cálculo de las computadoras -desde teléfonos móviles a superordenadores- crece de manera exponencial. Desde que la ley de Moore (66) fuera formulada estableciendo que el número de transistores en un circuito integrado se dobla cada aproximadamente 18 meses, este hecho no ha dejado de cumplirse, y según las declaraciones corporativas de Intel (67) seguirá siendo así al menos durante los próximos 10 años.

Durante mucho tiempo, este aumento de transistores y miniaturización de los componentes fue empleado en crear procesadores cada vez más complejos y rápidos, en la llamada "era de los MHz". Esta alcanzó su culmen en el 2004 con el Intel Pentium 4 "Presscott" a 3.8 GHz y la imposibilidad de producir una versión del mismo a 5.2 GHz comercialmente viable. Además de los problemas técnicos, la existencia de distintos cuellos de botella -acceso a disco, instrucciones de muy distinta velocidad de ejecución, dependencia de datos- hicieron abandonar el camino de los procesadores superescalares y supersegmentados con altísimas velocidades de CPU. Al contrario, la tendencia de desarrollo actual está centrada en procesadores más simples y lentos pero con varios núcleos, de manera que el rendimiento total obtenido sea mayor (68).

En este nuevo tipo de arquitecturas las aplicaciones deben estar preparadas para ser ejecutadas sobre varias unidades de proceso al mismo tiempo. Al igual que ocurre en los grandes equipos distribuidos, los códigos no paralelizables solamente aprovechan una unidad de computación, desaprovechando el resto del sistema. Y aunque existe una solución obvia -ejecutar varios de estos códigos al mismo tiempo, cada cual en un core/nodo- esto no es siempre posible ni conveniente. Más aun, con el crecimiento en número de CPU en los clúster, y el creciente número de núcleos por CPU, las aplicaciones no paralelas serán paulatinamente apartadas o reemplazadas por otras que sí puedan emplear de manera óptima los recursos disponibles.

Por otro lado, las tendencias actuales en diseño de equipos de alto rendimiento -y por extensión, en infraestructuras Grid- hacen que el tiempo de comunicación entre las tareas de una misma aplicación no sea constante, sino que dependa de la situación física de los elementos de proceso donde se estén ejecutando. Actualmente los posibles grados de separación son: dentro de un mismo chip, distintos chips en un mismo nodo, distintos nodos en un mismo clúster y distintos clúster en una misma organización virtual. El tiempo de

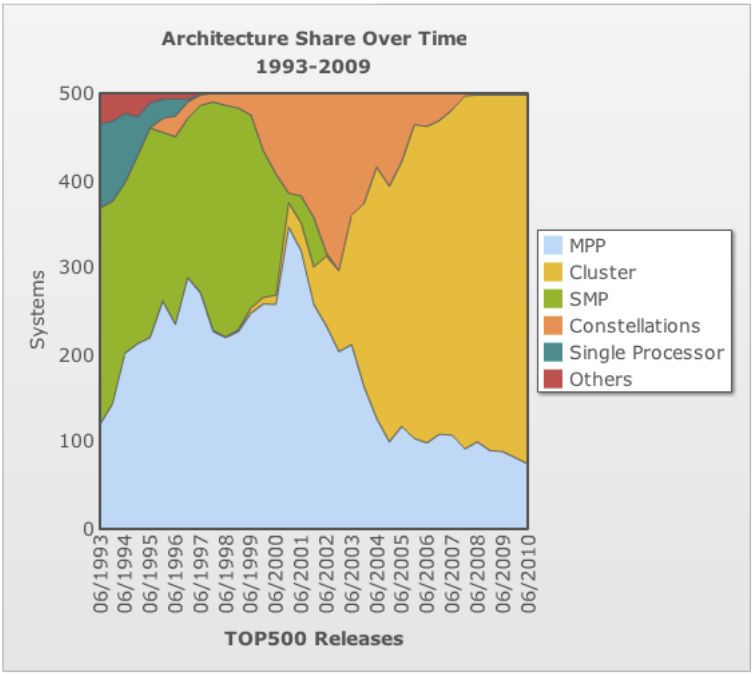

Figura 3.1: Tipos de arquitectura en el top 500 y su evolución temporal.

comunicación entre tareas oscila entre los milisegundos en el mejor de los casos y los minutos en el peor (cinco órdenes de magnitud) lo que convierte la creación de tareas masivamente paralelas en un reto. El diseño de estas aplicaciones no ha de centrarse únicamente en la paralelización del algoritmo de la manera más eficiente posible, sino que ha de tener también en cuenta esta variabilidad de las latencias para evitar pérdidas de rendimiento y limitaciones en la escalabilidad. Y como se verá en el siguiente apartado, los códigos de Monte Carlo representan una poderosa opción a tener en cuenta.

Todos los indicadores disponibles parecen señalar que esta tendencia al paralelismo masivo no es una moda pasajera o un producto de laboratorio y seguirá creciendo en el futuro, al menos a corto y medio plazo. Basten como ejemplos las figuras 3.1 y 3.2, mostrando la evolución en las arquitecturas y número de procesadores en los equipos del top500 a lo largo de su historia (69).

3.1.3. Futuro del código Monte Carlo

Teniendo en cuenta la arquitectura interna de los códigos de Monte Carlo, la información proporcionada hasta el momento debería ser suficiente para darse cuenta de su idoneidad para este tipo de recursos hardware. Dado que están compuestos por múltiples simulaciones completamente independientes,

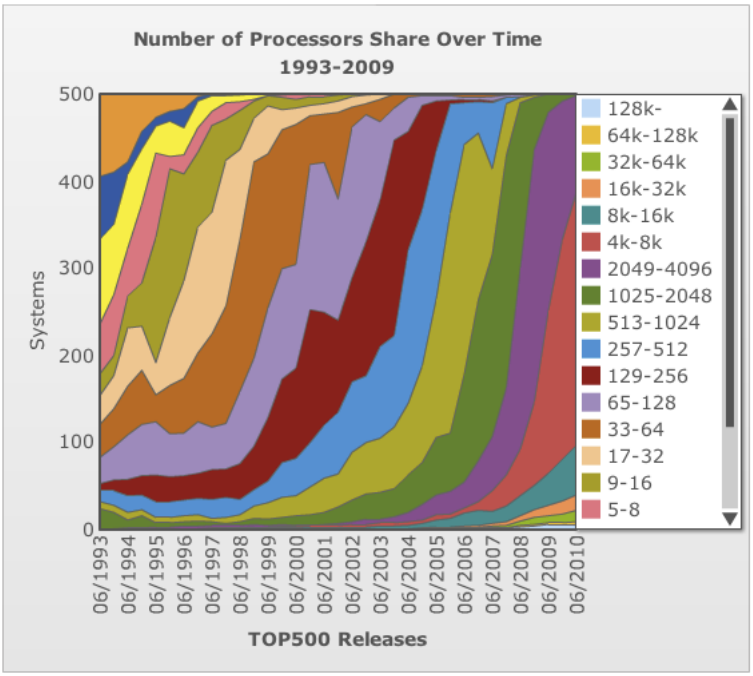

Figura 3.2: Evolución del número de CPU en el top500.

no es necesario que todas sean ejecutadas en el mismo recurso de manera secuencial ni que exista una comunicación continua entre ellas. Basta con ejecutar un número determinado de simulaciones en cada recurso, reunir los resultados de cada ejecución y analizarlos para obtener el resultado final. Esto permite superar todos los retos anteriormente planteados de una manera eficiente, sencilla y robusta. Así pues, el método de MC es un paradigma especialmente indicado para la computación paralela, por lo que previsiblemente seguirá siendo enormemente empleado en el futuro inmediato.

Algunos de los proyectos más grandes de la actualidad, tales como el LHC, están empleando códigos de MC de manera masiva (57). Así pues, su utilización al corto y medio plazo parece asegurada.

3.2. Arquitectura

Los códigos de Monte Carlo tradicionales están basados en la simulación de un número arbitrario de muestras independientes. Para hacer eso, típicamente están divididos en tres secciones. La primera lleva a cabo las operaciones comunes al principio de la ejecución: control de los datos de entrada e inicialización de las estructuras de datos; la segunda es la simulación del número deseado de muestras; y la tercera se encarga de computar

el resultado final a partir de los datos de las simulaciones anteriores.

Otro tipo de códigos de Monte Carlo, tales como los Random Walks (70) o las cadenas de Markov (71; 72) están fuera del objetivo de esta Tesis, y su estudio y modelamiento se propone como trabajo futuro al final de la misma.

3.2.1. Base matemática

J.M. Harmmersley y D.C. Handscomb (73) definieron el Método Crudo de Monte Carlo empleando un estimador insesgado y error estándar de la manera siguiente:

Si X_1,... X_n son números aleatorios independientes elegidos en una distribución entre 0 y 1, entonces las cantidades $f_i = f(X_i)$ son independientes y varían con una esperanza θ. Por tanto,

$$f' = 1/n \sum_{i=i}^{f_i} \tag{3.1}$$

es un estimador no sesgado de *sigma*. Si consideramos f' como el estimador crudo de θ, su varianza es

$$1/n \int_0^1 (f(x) - \theta)^2 dx = \sigma^2/n \tag{3.2}$$

Por tanto, el error estándar de f' es

$$\sigma_{f'} = \sigma/\sqrt{n} \tag{3.3}$$

mostrando que el error estándar de un análisis de Monte Carlo decrece con la raíz cuadrada del tamaño de la muestra.

Para reducir el error de un análisis de Monte Carlo, además de incrementar el tamaño de la muestra es posible aplicar técnicas de reducción de la varianza. Estas técnicas incorporan información adicional sobre el análisis directamente en el estimador, lo que permite hacer estimadores más deterministas y reducir el error estándar. Un estudio en profundidad de estas técnicas cae fuera del ámbito de la presente tesis doctoral. En cualquier caso, el trabajo de Liu (74) constituye un buen punto de entrada al área.

3.2.2. Implementación

La creación de una aplicación distribuida que emplee el método de Monte Carlo es relativamente sencilla, y puede llevarse a cabo siguiendo una estructura conocida. Considerando la arquitectura descrita en el apartado anterior, la figura 3.3 muestra un esquema con la arquitectura de la solución propuesta.

Como puede observarse, la arquitectura de las aplicaciones Monte Carlo anteriormente propuesta lleva naturalmente a un diseño en el que la implementación de un problema concreto se divide en tres secciones diferenciadas.

La primera, ejecutada en el recurso local, determina la manera en la que la simulación será llevada a cabo. En esta fase se decide el número y situación de los recursos de cómputo que se emplearán, así como los *samples* a ejecutar en cada uno. A continuación las tareas son ejecutadas en cada uno de los recursos remotos, simulando el número de *samples* deseado y devolviendo los resultados al recurso local. Por último, el recurso local se encarga de procesar todos los resultados parciales para obtener el resultado final.

Es posible emplear este esquema tanto en aplicaciones diseñadas para recursos locales (memoria compartida o clústers) como para infraestructuras Grid. En el primer caso se puede emplear un mecanismo de paso de mensajes para crear las tareas y que se comuniquen entre si. En este caso se emplea un sólo ejecutable: una tarea maestro se encarga de gestionar la ejecución (incluyendo la primera y tercera secciones) y los esclavos únicamente ejecutan la segunda. En el caso de infraestructuras Grid, todas las tareas son independientes y se ejecutan en distintas máquinas. Además, cada una de las secciones de la aplicación es llevada a cabo por un ejecutable diferente.

Puede concluirse que, gracias a sus características inherentes y la arquitectura propuesta, es posible ejecutar aplicaciones Monte Carlo en infraestructuras distribuidas. En las siguientes secciones se propondrán herramientas que hagan esta ejecución lo más eficiente posible, de manera que el consumo de recursos sea mínimo y el usuario pueda llevar a cabo sus simulaciones en el menor tiempo posible.

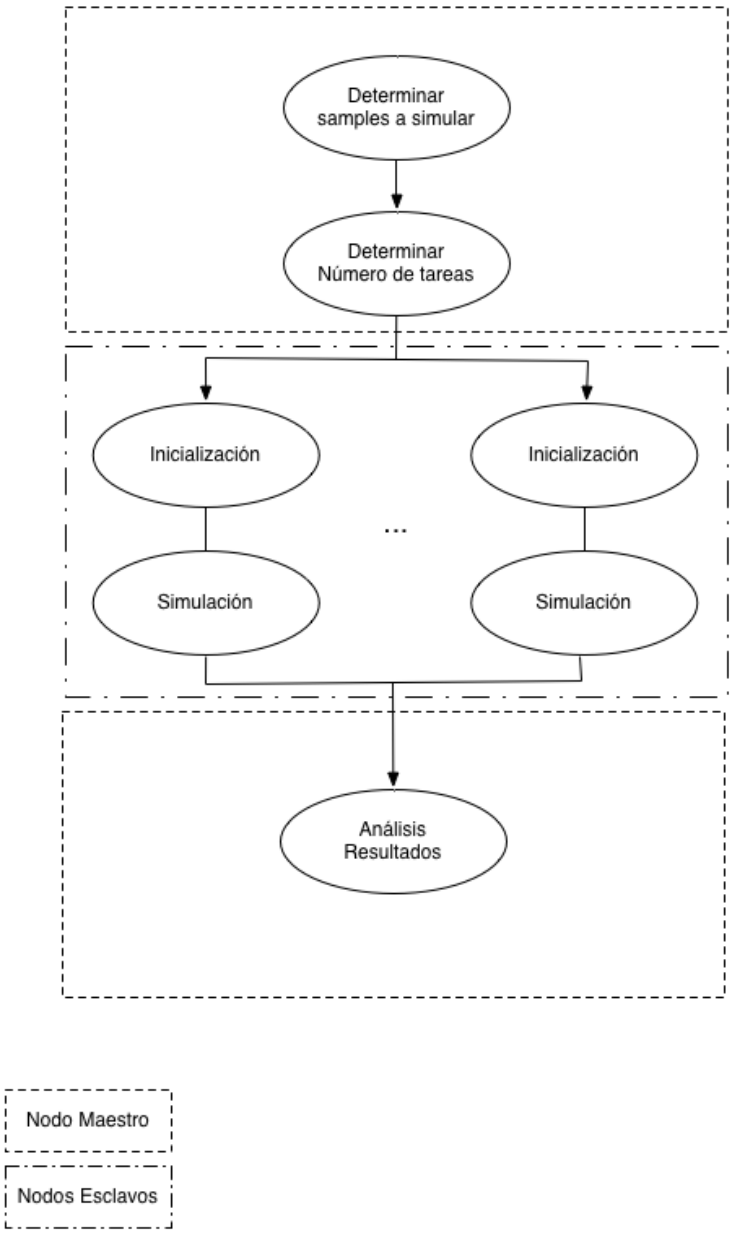

Figura 3.3: Diseño general de una aplicación Monte Carlo distribuida.

Capítulo 4

Planificación de iteraciones en Grid

En la siguiente sección se expondrá el algoritmo DyTSS, que constituye el núcleo de la propuesta intelectual de la presente tesis doctoral.

Para ello, el primer paso será proporcionar una visión general del *scheduling* en Grid. A continuación se describirá el *scheduling* de tareas independientes para reducir el *makespan*, ya que este es el objetivo a alcanzar. Después se proporcionará una visión general a la planificación de iteraciones y su aplicación a *scheduling* de Monte Carlo en Grid. Por último, una vez que se han detallado las bases teóricas en las que se asienta, el algoritmo DyTSS será propuesto.

4.1. *Scheduling*

En el ámbito de la computación distribuida, el *scheduling* puede definirse como la asociación de tareas a elementos de cómputo con el propósito de que sean ejecutadas.

Es bien conocido que la complejidad de un problema de *scheduling* general es NP-completo (75), es decir, que no puede ser resuelto en un tiempo polinomial. Además, como se mencionó en las secciones relativas a la arquitectura de infraestructuras Grid, el *scheduling* en este tipo de sistemas presenta complejidades adicionales. Afortunadamente, este es un campo ampliamente estudiado en el que se han obtenido numerosos avances y una eficiencia creciente.

Casavant *et al* (76) proponen una clasificación jerárquica sobre los algoritmos de *scheduling* en sistemas paralelos y distribuidos de propósito general. Dado que la Grid es uno de estos sistemas, los algoritmos aquí estudiados pueden ser encuadrados en esta taxonomía. Una representación esquemática de la misma se muestra en la figura 4.1 (77), donde las ramas cubiertas por

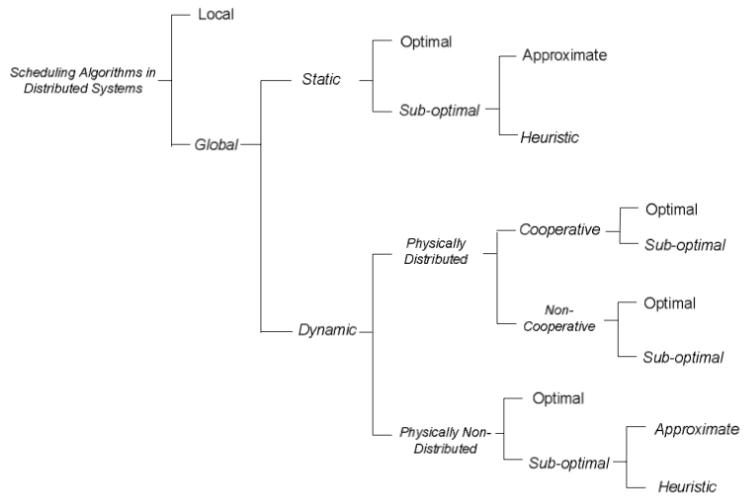

Figura 4.1: Taxonomía jerárquica de los algoritmos de *scheduling*.

los algoritmos de *scheduling* para Grid se muestran en cursiva.

El llevar a cabo una descripción completa del área cae fuera de los objetivos de este trabajo. Si se desea una referencia general -y excelente introducción a su estudio- los trabajos de Dong (77) y Xhafa (78) pueden ser un buen punto de partida.

En cualquier caso, a la hora de elegir el *scheduling* más adecuado para una aplicación concreta es necesario considerar el objetivo final que se desea conseguir. Dicho objetivo determinará el modo de planificación, así como la manera más eficiente de aprovechar los recursos Grid disponibles.

En el caso de trabajos con un gran volumen de datos -las llamadas *Data Intensive Applications*, Aplicaciones Intensivas en Datos- el esfuerzo principal ha de hacerse en una adecuada localización física de dichos datos, la minimización de la información transmitida a cada elemento de cómputo y la organización de la entrada y salida limitando los cuellos de botella. Así, Peris (79) propone una modificación del metaplanificador GridWay, creando un prototipo capaz de tener en cuenta las necesidades de datos a la hora de decidir el *site* idóneo para una tarea. Por su parte, Yu (80) introduce una familia de algoritmos llamada DLS (*Divisible Load Scheduling*, Planificación con Carga Divisible), adecuada para aplicaciones intensivas en datos. Dentro de esta familia de algoritmos es posible obtener particiones idóneas de los datos de entrada en tareas que se ejecutarán en entornos heterogéneos, al menos en entornos ideales y controlados.

Otras aproximaciones se centran en la ejecución de flujos de trabajo, esto es, tareas cuyos datos de entrada dependen de otras tareas. Esto establece

un orden en la ejecución y limita las posibilidades de planificación, con lo que el enfoque para mejorar su eficiencia es radicalmente distinto. En este ámbito cabe destacar el trabajo de Deelman (81), donde se analizan los modelos clásicos de *workflows* y su ejecución en Grid. Por otro lado un enfoque similar al de los *pipelines* en diseño de procesadores se emplea en el trabajo de González (82), dividiendo cada fase en "productor", "función computacional" y "consumidor". Resulta especialmente interesante ver cómo extrapola las técnicas de diseño de procesadores en computación Grid, de una manera parecida a lo que se realizará más adelante con las técnicas de loop-scheduling. En cualquier caso, los trabajos aquí expuestos corresponden a una pequeña muestra de un área en continua evolución.

Trabajos como el de Nguyen (83) muestran una mayor ambición, proponiendo un nuevo lenguaje de programación para HPC en Grid, ParoC++. Este lenguaje permite crear flujos de trabajo o aplicaciones intensivas en datos de una manera natural, integrando Globus para su ejecución en Grid.

En el caso de tareas independientes, es posible seguir múltiples aproximaciones. Una línea de trabajo como la seguida por Sakar (84) se centra en conseguir la mayor eficiencia posible de cara a cumplir con SLAs (*Service Level Agreements*, Acuerdos sobre el Nivel de Servicios). Otros enfoques, como Li (85), proponen un *scheduling* basado en QoS (*Quality of Service*, Calidad del Servicio), concretamente en los criterios de pago, *deadline* y fiabilidad.

Y por último, existen multitud de estudios como el presente, en los que el objetivo final del *scheduling* es reducir el *makespan*. En los siguientes apartados se analizarán diferentes alternativas de una manera más profunda, con el objetivo de mostrar aquellos puntos que no han considerado y posibles mejoras, justificando así la razón de ser de este trabajo.

Por supuesto, esta lista anterior no pretende ser intensiva. Su objetivo es únicamente mostrar la amplitud del campo de estudio, encuadrando el trabajo realizado dentro de su área y mostrando que la planificación de tareas en infraestructuras Grid es un campo enormemente activo donde distintas necesidades han llevado a enfoques diferentes.

4.1.1. *Scheduling* de tareas independientes para minimizar el *makespan*

Uno de los casos de uso más habituales de una infraestructura Grid es el de un usuario que desea ejecutar un número determinado de tareas en el menor tiempo posible. Este tiempo, medido desde que el usuario da la orden de ejecutar las tareas hasta que la última de ellas finaliza, es lo que habitualmente se denomina *makespan*.

4.1.1.1. Replicación de tareas

Dado que cada tarea a ejecutar es completamente independiente, la replicación de tareas representa una poderosa herramienta para reducir el tiempo de ejecución. Con esta aproximación el planificador comienza la ejecución de más tareas de las que serían estrictamente necesarias. Después, cuando el número deseado de simulaciones ha sido ejecutado, las tareas no finalizadas son abortadas. De este modo, es posible el evitar cuellos de botella que dan lugar a pérdidas de rendimiento o el fallo de un recurso en concreto, minimizando así el *makespan*.

Esta técnica ha sido ampliamente estudiada dentro del marco de la computación Grid. Li y Mascagni (86; 87) propusieron la llamada "*N-out-of-M strategy*" (estrategia del "N de M"). Para simular N tareas se envían M a la Grid, siendo $M > N$. La proporción M/N se obtiene a través de la tasa media de finalización de tareas en el sistema. Ambos artículos se centran en códigos de Monte Carlo y proporcionan una aproximación interesante. Sin embargo, parte de la complejidad de las infraestructuras Grid es obviada. La tasa de trabajos completados es global a la Grid -no a cada *site* y sin considerar que la Grid es dinámica- y los *overheads* relacionados con la transmisión de datos y el tiempo de cola no son incluidos. Además, el tamaño de las tareas enviadas a cada sitio es constante, por lo que no se aprovecha de la flexibilidad de los códigos de MC.

Silva (88) aplicó el concepto de "*bag of tasks*" ("bolsa de tareas") a los códigos de Monte Carlo, también en términos de replicación de tareas. En este caso se propone un algoritmo sin conocimiento del entorno llamado WQR (*Workqueue with Replication*, Colas de Trabajo con Replicación). WQR mejora la ejecución de los códigos de MC independientemente del estatus de la Grid en el momento del envío del trabajo. En este caso el tamaño de cada tarea enviada a la Grid es constante. Además, dado que la Grid es considerada una "caja negra", es menos eficiente que la aproximación seguida en este trabajo, donde se aprovecha toda la información posible que se puede obtener de cada *site*, tanto la pública como la obtenida del análisis de ejecuciones pasadas.

Vázquez-Poletti (89) estableció una aproximación similar, aunque centrada en la ejecución de flujos de trabajo. En su caso cada tarea es replicada tres veces, cancelando dos cuando una de ellas termina. Dado que su trabajo se centraba en la ejecución de flujos de trabajos, los resultados obtenidos no pueden extrapolarse directamente a este trabajo. Además, debido a que el número de réplicas está fijado antes de la ejecución de la aplicación y no basado en criterios objetivos y mensurables (más allá de la habilidad y experiencia del autor) da lugar a posibles mejoras.

Los resultados obtenidos con la aplicación de estas técnicas son altamente variables y dependen del tamaño del problema, el número de tareas a ejecutar y el estatus de la infraestructura Grid empleada para llevar a cabo

las mediciones. Mascagni (87) obtuvo una ganancia del 200 % con una proporción M/N de 2/1 -eficiencia perfecta-, estancándose en el 300 % cuando el número de réplicas tiende a infinito. En este caso no se llevó a cabo un análisis de los *overheads* producidos por la replicación, por lo que estos datos pueden variar ampliamente repitiendo el análisis con infraestructuras en producción. Silva (88) mejoró el rendimiento entre un 10 % y un 300 % en el peor y mejor caso, con el porcentaje de CPU desperdiciada oscilando entre el 40 % y el 105 %. Finalmente la aproximación de Vázquez-Poletti (89) obtuvo un aumento del rendimiento del 50 %, aunque en este caso el grado máximo de paralelismo estaba limitado por la estructura del *workflow*. Es importante tener en cuenta que el trabajo de Vázquez-Poletti es el único llevado a cabo en una infraestructura real, mientras que el resto únicamente proporcionan los resultados obtenidos en simuladores.

4.1.1.2. Agrupamiento de tareas

El agrupamiento de tareas es una técnica de control ampliamente utilizada en diferentes ramas del conocimiento y organización industrial (90).

En el caso de planificación de tareas en Grid, básicamente consiste en unir varias tareas en un solo trabajo llamado *chunk*, con el objetivo de minimizar algunos de los *overheads* producidos por el modelo de ejecución en Grid: el ejecutable es enviado al *site* remoto una sola vez -reduciendo el tiempo de transmisión- y el tiempo de cola es esperado sólo una vez por *chunk* en lugar de una vez por tarea. Esta aproximación surgió con la planificación de bucles en entornos heterogéneos distribuidos, llamada "loop self scheduling". frece una serie de ventajas que han hecho de esta técnica una herramienta muy extendida.

Como se ha mencionado previamente, los códigos de MC están constituidos por secciones fijas ejecutadas al principio y final de la ejecución y la simulación de un número arbitrario de *samples*. Si se tienen en cuenta los *overheads* antes mencionados y dado que son independientes del tamaño de las tareas, tiene sentido el llevar a cabo más de una simulación en cada instancia de la aplicación. El agrupamiento de tareas puede ser directamente extrapolado a los códigos de MC, considerando que el tamaño del *chunk* corresponde con el número de *samples* ejecutados en una tarea.

El trabajo de Choronopoulos *et al.* (91) presenta un concienzudo análisis de los algoritmos clásicos de *self-scheduling*. Herrera (92) propone una adaptación de varios de estos algoritmos a la computación Grid, empleando una arquitectura *master-slave*. En su tesis doctoral (93), propone además un nuevo algoritmo de *self-scheduling* denominado GTSS (*Grid Trapezoidal Self-Scheduler*, Auto-Planificador Trapezoidal para Grid).

Además de las técnicas clásicas, se han propuesto diferentes aproximaciones al problema en los últimos años. Entre las más destacables cabe mencionar: el empleo de un *scheduling* en dos niveles para adaptar la ejecución

a entornos dinámicos (94); el empleo de inteligencia artificial para elegir el mejor algoritmo de *self-scheduling* clásico para cada entorno (95); el uso de una base de datos con el rendimiento de todos los recursos de la infraestructura para poder modelar mejor su comportamiento (96); y, el empleo de una distribución cuadrática de tareas para adaptarse a un entorno heterogéneo tanto como sea posible (97).

Un punto débil común de estas técnicas es que no se considera la replicación de tareas. Además, llevar a cabo la distribución de tareas entre los diferentes recursos al principio de la ejecución no permite la adaptación de la ejecución a los cambios en la infraestructura que previsiblemente ocurrirán, tales como la presencia de nuevos *sites*, un número variable de recursos disponibles, pérdidas de rendimiento o errores de ejecución.

Observando el rendimiento de estas propuestas, en el trabajo de Jang (96) se muestra una mejora del 20 % comparada con el mejor algoritmo clásico y del 80 % con el peor. El resto de los trabajos (94; 95; 97; 98) ofrecen resultados muy distintos dependiendo del problema considerado y la infraestructura Grid, pero sus mejoras permanecen en el mismo orden de magnitud. Todos los experimentos mencionados menos el de Sun (98) fueron llevados a cabo en entornos controlados.

La bibliografía demuestra que el agrupamiento de tareas constituye una alternativa válida, al producir una mejora significativa del tiempo de ejecución simplemente optimizando la distribución de tareas. Sin embargo, en la mayoría de los trabajos existentes se dejan de lado ciertos aspectos relacionados con un entorno Grid real o incluso las pruebas no han sido ejecutadas en entornos de producción.

4.1.1.3. Otras aproximaciones

Por supuesto no es posible reducir las técnicas de *scheduling* a dos grupos, la duplicación y replicación de tareas. Este es un área enorme y han sido empleadas múltiples aproximaciones a lo largo del tiempo.

Como es sabido, los algoritmos genéticos son ampliamente empleados en el análisis de problemas NP completos. El caso de *scheduling* no es diferente y este tipo de algoritmos ha sido aplicado con éxito en multitud de ocasiones, tanto en sistemas locales (99) como en Grid (100) (101). Sin embargo este tipo de aproximaciones emplean un tiempo de computación no despreciable y en general su eficiencia en entornos dinámicos es menor que con las alternativas arriba detalladas.

Otros trabajos (102) proponen la creación de un *framework* en el que desarrollar aplicaciones, de manera que el resultado final sea un código ya paralelizado y capaz de ejecutarse en múltiples plataformas de manera eficiente. Ramírez (103) lleva a cabo un *scheduling* basado en las estimaciones del usuario sobre el tiempo de ejecución de tareas.

En general, puede deducirse que el gran problema del *scheduling* es la obtención de información fiable acerca de los recursos disponibles y la aplicación a ejecutar. Como se ha expuesto los disintos trabajos intentan solventarlo de maneras muy diferentes, o bien obviarlos al simplificar en exceso los entornos de pruebas donde analizan sus propuestas.

4.2. Planificación de iteraciones

Desde el comienzo de la computación moderna, los bucles y saltos han sido siempre analizados desde puntos de vista tanto académicos como funcionales. Baste citar el clásico trabajo de Dijkstra (104), uno de los padres de la computación moderna, proponiendo el uso de bucles y otras técnicas de control de flujo en lugar del `GOTO` para obtener un código de mejor calidad. Por la naturaleza misma de la computación, los bucles y las instrucciones condicionales (`IF/ELSE`) pueden ser considerados las herramientas más importantes que un programador tiene a su disposición, especialmente en programación iterativa. Así pues, han sido objetivo de análisis y optimizaciones desde su creación.

Como es sabido, es posible que existan dependencias entre las iteraciones de un bucle. Dichas dependencias pueden ser de control (si la ejecución de una instrucción depende del resultado de otra anterior) o de datos (los datos de entrada de una instrucción son los de salida de otra anterior). Estas dependencias limitan el paralelismo y la posibilidad de reordenación de instrucciones.

En los microprocesadores actuales existe más de una unidad de proceso. Es posible ejecutar varias instrucciones al mismo tiempo, siempre que no existan dependencias entre ellas. De este modo es posible acelerar el flujo de ejecución y enmascarar los diferentes cuellos de botella que existen dentro de un computador.

El análisis de dependencias pretende localizar estas relaciones entre instrucciones para generar restricciones en el orden de ejecución de las mismas. De no existir, pueden ser ejecutadas en el orden que se desee. En particular, las iteraciones de un bucle pasan a ser consideradas grupos de instrucciones independientes, que pueden ser ejecutados en el orden y elemento de cómputo que se desee -caso de que exista más de uno. éste es el punto de partida de los algoritmos de planificación de iteraciones.

Con la aparición de las arquitecturas distribuidas, y en concreto la computación Grid, surgió la idea de aplicar la planificación de iteraciones al *scheduling* de tareas independientes. Después de todo, las similitudes son muchas:

- Se trata de grupos de instrucciones que pueden ser tratados de manera independiente (iteraciones de un bucle versus instancias de una aplicación).

- Ejecutándose en elementos de cómputo distribuidos (diferentes elementos de un procesador versus diferentes equipos).
- Con el propósito de minimizar el tiempo de ejecución enmascarando los *overheads* asociados (acceso a memoria versus copia a *sites* remotos).

Esta aproximación permite obtener una gran eficiencia en la ejecución de tareas en Grid, aun siendo algoritmos relativamente simples y que no necesitan software adicional en los *sites* remotos. Por tanto, constituye una poderosa alternativa a la hora de planificar tareas y es la empleada en el algoritmo propuesto en la presente Tesis Doctoral.

Con el propósito de proporcionar la base teórica sobre la que se asienta el trabajo realizado, a continuación se incluye una descripción de los principales algoritmos de planificación de bucles. Esta recopilación ha sido obtenida de (91), ampliada posteriormente por Herrera en su tesis doctoral (93).

4.2.1. Algoritmos estáticos

En los algoritmos estáticos el cálculo del número de trabajos a ejecutar en cada nodo se realiza en tiempo de compilación.

Por su naturaleza, este tipo de algoritmos son los menos flexibles: es necesario conocer el número de tareas y procesadores antes de compilar la aplicación, tarea imposible en una infraestructura dinámica como es el Grid.

Como ejemplos de este tipo de algoritmos se pueden citar el Planificador por Bloques (PB) (105) y Planificador Cíclico (PS) (105). Para un número de tareas N y un número de procesadores P ambos asignan $\lceil N/P \rceil$ tareas a cada procesador, siendo las de PB consecutivas y las de PS cíclicas con periodo P.

4.2.2. Algoritmos dinámicos

Los algoritmos dinámicos suponen una importante evolución respecto a los algoritmos estáticos. En este caso la distribución de tareas se realiza en tiempo de ejecución y no de compilación, lo que permite el ejecutar aplicaciones en las que no se conoce su tamaño de antemano.

En este tipo de algoritmos, el número de tareas a ejecutar en cada recurso -originariamente, número de iteraciones del bucle- es denominado *chunk*. El número y tamaño del *chunk* a ejecutar en cada recurso es lo que diferencia los diferentes algoritmos.

Llevando al extremos este tipo de algoritmos se encuentra la posibilidad de ejecutar una única tarea por *chunk*, en el llamado PSS (*Pure Self Scheduling*, Auto Planificador Puro) (106). En el caso del CSS (*Chunk Self Scheduling*, Auto Planificador de *Chunks*) es el usuario el que decide el tamaño de cada *chunk*. Y en GSS (*Guided Self Scheduling*, Auto Planificación

Guiada) (107) se incluye además una función de decrecimiento, con lo que el tamaño de los *chunks* decrece en el tiempo.

4.2.3. Algoritmo TSS

El algoritmo TSS (*Trapezoidal Self-Scheduling*, Auto Planificación Trapezoidal) (108) es una combinación del GSS y CSS, obteniendo lo mejor de ambos. Por su diseño es capaz de mantener una carga equitativa entre todos los procesadores (como el GSS) a la vez que su coste de planificación es lineal (como el CSS).

En el algoritmo TSS, el número de tareas C_i para cada *chunk* i se define como

$$C_i = C_{i-1} - D \; con \; D = \lfloor \frac{(F-L)}{(N-1)} \rfloor \; y \; N = \lceil \frac{(2I)}{(F+L)} \rceil \; donde \; C_0 = F \quad (4.1)$$

En este caso los parámetros F y L (de First y Last, Primero y Último) pueden ser definidos por el usuario o calculados mediante la siguiente función:

$$F = I/2p \; y \; L = 1 \quad (4.2)$$

Como muestra la ecuación 4.1 en el algoritmo TSS sólo es necesario calcular el factor de decrecimiento D una vez por ejecución, lo que mantiene la complejidad del algoritmo reducida.

4.2.4. Algoritmo GTSS

El algoritmo llamado GTSS (*Grid Trapezoidal Self-Scheduler* Auto-Planificador Trapezoidal para Grid) es una modificación del algoritmo TSS (*Trapezoidal Self-Scheduling*, Auto-Planificador Trapezoidal) (108), adaptado a infraestructuras Grid. GTSS está basado en tres elementos principales: un modelo de *benchmark* para Grid; el factor de relación R^i_{min}; y, el algoritmo dinámico TSS. Fue propuesto por Herrera en su Tesis Doctoral (93).

4.2.5. Rendimiento de una infraestructura Grid

El rendimiento de una infraestructura Grid puede ser modelado con sólo dos parámetros (109; 110), el *asymptotic performance* y el *half performance length*.

- *Asymptotic performance* r_∞ es el máximo rendimiento de la infraestructura, medido en tareas completadas por segundo. En caso de que un array homogéneo de N procesadores con un tiempo de ejecución de T segundos por tarea, su valor es de N/T.

- *Half performance length* $n_{1/2}$ es el número de tareas requeridas para obtener la mitad del rendimiento asintótico. Este parámetro mide el grado de paralelismo del sistema visto desde la aplicación. En el caso de una aplicación mavisamente distribuida su valor es de $N/2$.

4.2.5.1. Descripción del algoritmo GTSS

GTSS emplea el modelo de *benchmark* para Grid anteriormente descrito, utilizando r_∞ y $n_{1/2}$ para ajustar el tamaño del *chunk*.

El factor de heterogeneidad R^i_{min} es un coeficiente que relaciona el tiempo de ejecución de cada tarea T^i_{wall} de cada nodo en una infraestructura Grid. Se define del modo siguiente:

$$R^i_{min} = T_{min}/T^i_{wall} \tag{4.3}$$

Siendo T_{min} el menor de todos los tiempos de ejecución.

Con esto, el comportamiento del algoritmo GTSS para un nodo determinado j se define como:

$$C^0_j = \lceil F * R^j_{min} \rceil \ siendo\ F = I/4n_{1/2},\ L = 1 \tag{4.4}$$

$$C^i_j = C^{i-1}_j - D\ siendo\ D = \lfloor (F-L)/(N-1) \rfloor\ ,\ N = \lceil 2I/(F+L) \rceil \tag{4.5}$$

Aquí F es el tamaño del primer *chunk*, L es el tamaño del último, N el número de pasos e I el número de tareas a ejecutar. F e I pueden ser fijados estáticamente o emplear la Ecuación 4.4 para obtener su valor.

4.3. Planificación de iteraciones y códigos de Monte Carlo

Para la ejecución en Grid de los códigos de Monte Carlo, es posible apoyarse en que todas las simulaciones son igualmente útiles para la obtención del resultado final.

La forma habitual de trabajar no es en principio diferente a la empleada con otro tipo de aplicaciones. Primero, un determinado número de tareas es enviado a la Grid para llevar a cabo las simulaciones necesarias. Previsiblemente algunas de ellas fallarán o proporcionarán un resultado erróneo, con lo que tendrán que ser enviadas de nuevo hasta cumplir con su cometido. Cuando se alcanza el número necesario de simulaciones, sus resultados parciales son computados para obtener el final.

En el caso de los códigos de Monte Carlo, su naturaleza hace que cada una de esas simulaciones sea completamente independiente de las demás. Esto quiere decir que no existen dependencias de control o datos que impongan

un orden en su ejecución. Por tanto, para su empleo en Grid es posible utilizar cualquier técnica de *scheduling* para tareas independientes, ya que el número, tamaño y lugar de ejecución de cada tarea depende únicamente de los deseos del planificador y del estado de la infraestructura, no de factores relativos a la aplicación.

Como se razonó en la sección 3.2, los códigos de MC están constituidos por secciones fijas al principio y al final de la ejecución y la simulación de un número arbitrario de muestras. Si se tienen además en cuenta los *overheads* asociados a la ejecución de cualquier tarea en una infraestructura Grid, y dado que son independientes de la duración de dicha tarea, tiene sentido el realizar más de una simulación en cada instancia de la aplicación.

Por último, el hecho clave es que en el caso de los códigos de Monte Carlo cada muestra es inicializada con un número aleatorio. Esto evita los problemas habituales de la replicación de tareas: en este caso no es necesario desechar una tarea dado que otra réplica ya ha sido ejecutada -con la pérdida de tiempo y capacidad de cálculo que esto implica- sino que todas son igualmente válidas para calcular el resultado final. Este hecho, proporciona una enorme flexibilidad añadida que es posible aprovechar en el *scheduling*.

Con esto en mente, la replicación de tareas puede entenderse de dos maneras: enviando a la Grid más tareas de las necesarias para llevar a cabo el experimento deseado o enviar las mismas, pero cada una con un número mayor de simulaciones por ejecutar. Así mismo, el agrupamiento de tareas puede ser extrapolado a los códigos de MC considerando que el tamaño de un *chunk* se corresponde con el número de *samples* ejecutados en una tarea. Igualmente, la planificación de iteraciones se extrapola directamente a los códigos de Monte Carlo suponiendo que un determinado número de tareas independientes ejecutadas en un *site* corresponde a ese mismo número de *samples* de la aplicación Monte Carlo.

4.4. Propuesta: DyTSS

En este trabajo se presenta un nuevo algoritmo de *scheduling* llamado DyTSS (*Dynamic Trapezoidal Self Scheduling*, Auto Planificación Trapezoidal Dinámica). Está basado en el algoritmo GTSS adaptado para ejecutar códigos de Monte Carlo en infraestructuras extremadamente dinámicas.

4.4.1. Problemas con los algoritmos de *self-scheduling* puros en Grid

A lo largo del desarrollo de este trabajo, la ejecución de algoritmos de *self-scheduling* en Grid fue ampliamente estudiada. En ese análisis, se detectaron una serie de problemas significativos.

El primero es que, en este tipo de algoritmos, la agrupación de tareas

en diferentes *chunks* es llevada a cabo al principio de la ejecución. Aunque esta aproximación es válida en entornos estáticos, es irrealizable en infraestructuras Grid, que son por definición dinámicas (2). El estado de cada *site* -número de *slots* libres, tiempo de cola y demás- se considera fijo a lo largo de la ejecución de la simulación, obviando así una característica básica de la Grid. Por otro lado, la relación propuesta entre *sites* y *chunks* no es dinámica, con lo que el fallo de un solo recurso resulta en una simulación inacabada.

Aunque los algoritmos clásicos de *self-scheduling* se comportan extremadamente bien en entornos controlados, esta falta de adaptación los hace subóptimos en entornos de producción. Aun más, la posibilidad de no acabar una simulación por el fallo de un solo recurso hace que estas aproximaciones no sean válidas para infraestructuras en estado de producción.

Para superar estos problemas se ha propuesto un nuevo algoritmo, DyTSS. La técnica y parámetros empleados para dividir la simulación en *chunks* es similar a la de GTSS, pero esta división no se lleva a cabo al principio de la ejecución sino cada vez que se crea un nuevo *chunk*. De ese modo, DyTSS es capaz de superar los problemas anteriormente mencionados y a la vez beneficiarse de la distribución irregular de *chunks* de los algoritmos tradicionales de *self-scheduling* para aprovechar al máximo un conjunto de recursos heterogéneos.

4.4.2. Algoritmo DyTSS

Como una particularidad de esta técnica de planificación, las fases de *Grid Characterization* y *Scheduling* se realizan al mismo tiempo debido a la fuerte relación que existe entre ellos. El algoritmo 1 muestra una implementación de alto nivel de las fases necesarias.

Primero, la información acerca de la infraestructura es obtenida tal y como ha sido explicado. Con esta información la simulación es dividida entre todos los recursos disponibles, asignando el mismo número de *samples* a cada uno.

A continuación se emplea el bucle mostrado en el algritmo 2 para ajustar esta distribución inicial de la manera siguiente:

- Se calcula el rendimiento de la infraestructura con la distribución actual de tareas.
- Se aplica el algoritmo GTSS para redistribuir la carga de trabajo entre todos los recursos.

Este bucle es repetido hasta que se alcanza una situación estacionaria, en la que no hay variaciones en la lista de tareas.

La complejidad del bucle es lineal con el número de recursos. De todos modos, dado que está compuesto únicamente por unas pocas operaciones

Algorithm 1 Descripción del algoritmo de *scheduling* DyTSS.

```
Require: desiredSamples > 0
  remainingSampeles = desiredSamples
  taskList = creare_task_list(remainingSamples)
  submit(taskList)
  while remainingSamples > 0 do
    for task in taskList do
      if task finished execution then
        remainingSamples = remainingSamples -
        task.get_number_of_samples()
        auxTaskList = create_task_list (remainingSamples)
        newTask = auxTaskList.pick_first_task(task.getResource())
        submit(newTask)
        lastList.add(newTask)
      end if
    end for
  end while
```

Algorithm 2 Descripción de la función *create_ task_ list()*.

```
Require: samples > 0
  sitestatus = read_site_status()
  taskList = create_simple_task_list(siteStatus)
  gridPerformance = calculate_performance(taskList)
  for i = 0 to 100 do
    oldTaskList = taskList
    taskList = DyTSS_get_task_list(gridPeformance,siteStatus)
    gridPerformance = calculate_performance(taskList)
    if taskList = oldTaskList then
      break
    end if
  end for
  return taskList
```

aritméticas por recurso su tiempo de ejecución es despreciable con el tamaño de las infraestructuras Grid actuales. Se ha establecido un límite de 100 iteraciones basado en la experiencia obtenida durante la creación del algoritmo: en el caso de *livelocks*, habitualmente se dan antes de la iteración número 10. Se ha establecido el límite en el siguiente orden de magnitud ya que, a pesar de que habitualmente no realice trabajo útil en estos casos, la proporción coste/beneficio es favorable a un límite alto que asegure que el algoritmo nunca se detendrá antes de un equilibrio o *livelock*.

Es importante señalar que en este algoritmo C_i^j se calcula siempre con la fórmula para C_0^j de la Ecuación 4.4. La reducción del ramaño de tarea viene dada por la reducción de I cada vez que ha acabado una tarea anterior. Por tanto, los pasos planteados en la Ecuación 4.5 no son necesarios aquí.

Hay otra planteamiento innovador en el proceso de *scheduling* propuesto. La aplicación del algoritmo de *scheduling* al principio de la ejecución no se emplea para obtener la distribución de tareas en toda la simulación, sino sólo una tarea para cada *slot* disponible. Después, cuando una tarea acaba o un nuevo recurso es descubierto, el algoritmo se ejecuta de nuevo, empleando información obtenida en tiempo real acerca de los *sites* para obtener el nuevo tamaño de tarea. De este modo, este tamaño de tarea es siempre calculado con la información más reciente, mejorando así la efectividad del algoritmo. Además, el fallo o disminución de rendimiento de cada recurso no representa un cuello de botella, como ocurre con los algoritmos tradicionales de *self-scheduling*.

Por último, merece la pena destacar una decisión técnica. El número de tareas enviadas a cada *site* no es el número de *slots* disponibles sino un 20 % mayor. Esto es debido a dos factores complementarios.

El primero es una reducción del tiempo de ejecución total. Si una tarea es enviada sólo después de que la anterior ha finalizado, se produce un overead debido a la necesidad de copiar los datos de salida al recurso local y comprobar que la ejecución fue correcta antes de lanzar la siguiente. Por otro lado, si la nueva tarea a ejecutar está ya encolada en el *site* remoto, empieza su ejecución nada más acabar la anterior, enmascarando el *overhead* descrito y reduciendo así el *makespan*.

En este punto se podrían utilizar técnicas de agrupamiento de tareas en lugar del envío de tareas de diferente tamaño. Aunque eso reduciría los tiempos de ejecución antes mencionados, representaría una pérdida de flexibilidad. Con esta aproximación el tamaño de las tareas es determinado dinámicamente por el estatus y rendimiento de la infraestructura Grid entera, lo que no podría ser hecho si todas las tareas que se envían a un *site* determinado fueran creadas a la vez y enviadas al comienzo de la ejecución.

La segunda razón es que el número de *slots* disponibles en cualquier *site* pueden variar a lo largo de la ejecución de la aplicación. Si es reducido, Montera lo detectará y reducirá el número de tareas enviadas, cancelando

aquellas que están en espera -de manera que no se pierde tiempo de CPU. Si es incrementado, algunas de las tareas que componen ese 20 % sobrante comenzarán su ejecución, Montera lo detectará, y creará más hasta alcanzar de nuevo ese 20 %. Con esta decisión de planificación se asegura que será ejecutado un número óptimo de tareas en cada recurso, manteniendo a la vez el número de réplicas reducido. En la sección de resultados puede verse la efectividad de esta técnica en un clúster en producción.

Es necesario señalar que las diferencias entre esta técnica y la replicación de tareas es sutil pero importante. En ambas aproximaciones el número de tareas enviadas es mayor de lo necesario, pero su uso es diferente. En el caso de la replicación se conserva la tarea que antes termine y se eliminan sus réplicas, perdiendo el esfuerzo computacional empleado en su ejecución. Pero en el caso de DyTSS, dado que en los códigos de MC todos los resultados son igual de válidos, los resultados de todas tareas son empleados en la simulación. De este modo DyTSS obtiene las ventajas de la replicación -mayor productividad y desaparición de los cuellos de botella- mientras que la infraestructura Grid empleada no sufre el *overhead* y abuso inherente a los algoritmos de replicación.

4.5. Conclusiones

El *scheduling* de tareas en Grid, a pesar de ser un campo ampliamente estudiado, sigue estando abierto a innovaciones y mejoras.

En la presente sección ha quedado demostrado que la planificación de iteraciones es una potente alternativa en el *scheduling* de este tipo de tareas. A su simplicidad y facilidad de implementación se le añade que no se necesita software específico. De este modo, únicamente mediante el empleo de una adecuada planificación se obtiene un rendimiento comparable o mayor al de otras alternativas más sofisticadas y que necesitan un enorme despliegue de medios. Este hecho facilita la tarea del usuario final, posibilitando el aumentar el rendimiento de sus aplicaciones de una manera sencilla y eficaz.

La alternativa propuesta, DyTSS, supone unir la potencia de la planificación de iteraciones con la flexibilidad de los códigos de Monte Carlo, obteniendo así lo mejor de ambas partes. Mediante una adecuada modelización de los recursos y la aplicación a ejecutar, es capaz de decidir en tiempo de ejecución el número y tamaño de cada tarea que será enviada a la Grid, adaptándose así a cambios en la infraestructura o problemas en alguno de los *sites* empleados.

Capítulo 5

Montera

En la siguiente sección se describe con detalle Montera, el *framework* implementado durante el desarrollo de esta Tesis.

El objetivo al implementar Montera es doble. Primero, con la creación de este *framework* se pretende demostrar de una manera práctica la superioridad del algoritmo de planificación antes propuesto frente al resto de las alternativas. Y tras cumplir este objetivo, se ha perfeccionado Montera y dotado de una interfaz sencilla, con el objetivo de que los investigadores que así lo deseen puedan utilizarlo para la ejecución de sus códigos de Monte Carlo.

5.1. Introducción

La computación Grid, por definición, no es un entorno controlado. En el capítulo correspondiente se ha explicado el dinamismo de este tipo de infraestructuras, así como su propensión a fallos y problemas de todo tipo. Por tanto, cualquier aplicación que vaya a ser ejecutada en Grid habrá de tener estos hechos en cuenta, y emplear las herramientas necesarias para que influyan lo menos posible en su tiempo de ejecución y la corrección del resultado obtenido.

Por otro lado, es importante tener en cuenta que cualquier código tiene características propias y distintivas. Estas características deben ser tenidas en cuenta a la hora de planificar su ejecución, buscando las posibles sinergias con la infraestructura en que será ejecutado. De este modo las particularidades de dicho código supondrán una ventaja y no un inconveniente, logrando un menor *makespan* y un aprovechamiento más eficiente de la infraestructura.

Hasta el momento se ha puesto un gran esfuerzo en la mejora del rendimiento de aplicaciones en Grid, con muchas aproximaciones distintas basadas en diferentes conceptos de "infraestructura Grid", las características particulares de la aplicación a ser ejecutada o la información disponible por

el planificador. En cualquier caso, la mayoría de estos desarrollos no contemplan una posible evolución de la infraestructura a lo largo del tiempo, no les afecta el código que se está ejecutando o han sido probados únicamente en entornos controlados o simuladores. Por ello, es necesario tener en cuenta todas estas condiciones a la hora de valorar la validez de una propuesta, con el fin fundamental de que los resultados obtenidos sean válidos en un entorno de producción y no únicamente a nivel de investigación.

Como se detalló en el capítulo 3, los códigos de MC son ampliamente empleados en la ciencia. De la física de altas energías a ciencias de la vida, de fusión a ingeniería, su capacidad de modelar fenómenos complejos proporciona buenas aproximaciones a muchos problemas donde encontrar una solución exacta no es realista. Además, su posible distribución en tareas independientes los hace idóneos para la computación Grid.

Esa es la razón por la que en este trabajo se considera necesaria la creación de un *framework* específico para la ejecución de códigos de MC en Grid, Montera. Este *framework* se apoya en el uso de tres herramientas complementarias destinadas a obtener información de diferentes fuentes, posteriormente empleada para distribuir los *samples* a ejecutar entre los diferentes recursos:

- El uso de información acerca de la infraestructura Grid basada en datos estáticos y dinámicos, obtenidos a partir de ejecuciones pasadas y el estatus actual.
- El análisis automático y la caracterización de la aplicación a ejecutar con el fin de modelar su comportamiento.
- El empleo del algoritmo DyTSS, que determina el tamaño de cada *chunk* en el momento de su ejecución; con ello, se adapta el envío de trabajos a los cambios en el número, características o tamaño de los recursos disponibles dentro de un entorno dinámico.

Así pues, el primer paso en la ejecución de un código empleando Montera consiste en la realización de un *profiling* del mismo para conocer sus características y la manera que funciona -ya que, como se ha explicado en la sección correspondiente, los códigos de MC poseen una estructura común.

Después, un mecanismo iterativo implementado en Montera que emplea el *benchmark* Whetstone (111) como unidad mide el rendimiento de los *sites* remotos.

Para la caracterización completa de los *sites* Grid son empleados todos los parámetros que es posible recolectar: información publicada; whetstones; tiempo de cola; ancho de banda; número de tareas ejecutadas correcta y erróneamente; etc. Cuando es necesario no se emplea sólo el valor medio, sino también la desviación típica.

El rendimiento de una infraestructura Grid se modela con los parámetros *asymptotic performance* y el *half performance length* (109; 110), aunque

teniendo en cuenta que las tareas son de tamaño variable.

Por otra parte, entre los algoritmos de planificación implementados en Montera se incluye el llamado DyTSS, que ha sido explicado en la sección correspondiente.

El *scheduling* es llevado a cabo en dos puntos distintos en lo que constituye una novedosa aproximación al área. En primer lugar, del mismo modo que en el resto de los planificadores existentes, es realizado en el recurso local determinando el tamaño del *chunk* a ejecutar antes de su envío al *site* remoto. Pero además -y aquí reside la innovación- es posible efectuar un ajuste del tamaño del *chunk* en el *site* remoto inmediatamente antes de su ejecución. De este modo, el *chunk* ajusta su tamaño dinámicamente dependiendo del estatus del recurso remoto y las necesidades del usuario.

Por último, una interfaz fácil de usar es también ofrecida al usuario junto a las herramientas necesarias para implementar diferentes políticas.

5.2. Modelado de códigos de Monte Carlo

Montera implementa el modelo de Monte Carlo que se expuso en el apartado teórico dedicado a este paradigma, brevemente descrito a continuación para facilitar la labor del lector.

Básicamente, los modelos de Monte Carlo están basados en la simulación de un número arbitrario de *samples* independientes. Para hacer esto, un código de MC típico se divide en tres secciones. La primera lleva a cabo las operaciones comunes al principio de la ejecución: control de los datos de entrada, inicialización de las estructuras de datos y el resto de operaciones necesarias; la segunda es la simulación del número deseado de muestras; y la tercera recopila los datos de las simulaciones anteriores y los analiza para computar el resultado final. La ejecución de cada sección depende del número de *samples* a simular y la primera y tercera pueden incorporar también un tiempo constante. Por tanto, la ejecución del código completo depende del número de *samples* más un tiempo constante.

En esta propuesta, los códigos de MC están representados por dos parámetros, *constant effort* y *sample effort*. *Constant effort* representa el esfuerzo necesario para ejecutar la parte constante de la aplicación, y *sample effort* es el esfuerzo necesario para simular un solo *sample*.

Los códigos de MC son intensivos en CPU (112; 113) y en muchos casos están dedicados a operaciones de punto flotante. Por eso, la unidad elegida para los parámetros propuestos es $segundos * whetstone$.

El *benchmark* Whetstone (111) es ampliamente empleado para estimar la velocidad de un recurso computacional al trabajar con puntos flotantes. La unidad en la que se describe el resultado, "miles de instrucciones de Whetstone por segundo", será acortada a whetstones desde aquí en adelante. Así pues, los whetstones de un *site* determinado y los dos parámetros menciona-

dos proporcionan suficiente información para estimar de una manera precisa el tiempo de ejecución de una instancia determinada del código en un recurso en particular.

Para obtener estos valores, un mecanismo de *profiling* del código ha sido incorporado en Montera. Este mecanismo consiste en el envío a uno o varios *sites* de un *script* que:

- Ejecuta el *benchmark* whetstone en el sito remoto para medir su rendimiento.
- Ejecuta la aplicación que queremos modelar para que los parámetros *constant effort* y *sample effort* puedan ser determinados. Esta ejecución puede durar un tiempo prefijado o el tiempo que el usuario considere necesario.

Al repetir este proceso en diferentes *sites*, la influencia de factores exógenos -tales como variaciones del rendimiento o imprecisiones del *benchmark*- es minimizada.

Por último, un análisis de los resultados basados en la Ecuación 5.1 es llevado a cabo.

$$Execution\ Time\ N\ Sample = C_e e + N * S_e(2) \tag{5.1}$$

Donde C_e corresponde a *constant effort*, S_e a *sample effort* y N es el número de *samples* simulados en una ejecución determinada. Como puede verse, los parámetros deseados se pueden obtener después de tan sólo dos ejecuciones del código. De todos modos, para mejorar la precisión de los resultados, es ejecutado varias veces con un valor de N creciente, y los parámetros son obtenidos como una media ponderada de los resultados parciales. Finalmente, estos valores son normalizados con el resultado de la ejecución del *benchmark*.

5.3. Análisis del rendimiento de los *sites* Grid

Al lidiar con el *scheduling* de tareas en infraestructuras Grid, son posibles -como se ha explicado previamente- dos aproximaciones diferentes: la infraestructura Grid puede ser considerada una caja negra o un conjunto de recursos conocidos. Aquí se emplea la segunda, por lo que que se ha realizado un gran esfuerzo en obtener tanta información como sea posible acerca de su composición.

Con ello, los *sites* han sido caracterizados de acuerdo a los siguientes parámetros:

- Whetstones: eficiencia al trabajar con números flotantes.

- Tiempo de cola: tiempo medio de cola para una tarea al ser ejecutada.
- Ancho de banda.
- Número de ejecuciones exitosas y fallidas en ejecuciones pasadas.
- *slots* disponibles en ejecuciones pasadas.
- Número de intentos fallidos a la hora de hacer el *profiling* del *site* cuando se llevó a cabo el último.

En el caso de los dos primeros parámetros no se almacena sólo el valor medio sino también la desviación típica. Esto es necesario por ser elementos con una gran variabilidad en determinados recursos, con lo que la media no es suficiente para representar su comportamiento de una manera realista.

Esta información incrementa el conocimiento que se posee sobre cada *site* comparado con las herramientas existentes, únicamente basadas en la información pública -arquitectura, MHz de su CPU y el resto. Esto representa una clara ventaja en el proceso de *scheduling*, ya que un mayor conocimiento de la infraestructura permite la toma de ejecuciones más precisas.

Montera incluye un mecanismo de *profiling* para los *sites* de la infraestructura Grid, empleado para obtener esta información.

Cuando un nuevo *site* es detectado o se detecta que uno conocido ha sido modificado -diferente CPU o memoria-, Montera ejecuta un *benchmark* para obtener los parámetros anteriormente mencionados.

Este proceso de *benchmarking* es fundamental para entender el rendimiento de la infraestructura. A pesar de que la primera vez que Montera es ejecutado representa una sobrecarga importante, la información es almacenada y recuperada cuando se necesita, por lo que esta sobrecarga desaparece a medio plazo. Montera también actualiza la información sobre los recursos después de cada ejecución, incrementando así el conocimiento sobre la infraestructura y mejorando el rendimiento de la aplicación, todo con un conste computacional despreciable.

El resultado del *benchmark* es también empleado para controlar la disponibilidad del *site*. Después de tres intentos fallidos, el *site* es marcado como problemático por un periodo determinado de tiempo -una semana por defecto, pero puede ser fácilmente modificado según el criterio del usuario- y excluido de la lista de recursos válidos. De este modo Montera se asegura que todas las tareas serán enviadas a recursos con un funcionamiento correcto, maximizando así las posibilidades de una ejecución exitosa.

El número de *slots* disponibles en ejecuciones pasadas es leído al principio de Montera para decidir el la cantidad de tareas que se ejecutarán así como su tamaño. Una vez que la ejecución ha comenzado y hay *chunks* siendo ejecutados en distintos *sites* este valor es actualizado en tiempo real, de

manera que cada política de *scheduling* puede emplear esta información para decidir cómo dividir el problema y dónde enviar cada *chunk*.

Es destacable que se ha utilizado la propuesta de Biessel (114) para calcular la media y desviación típica de un conjunto creciente de valores sin la necesidad de almacenar todos ellos. Dada la media y desviación típica de un conjunto de N elementos y un nuevo elemento n, los autores proponen un algoritmo para obtener la media y desviación típica de un conjunto compuesto por los $N+1$ elementos. De este modo no es necesario almacenar el conjunto entero de valores, y son necesarios menos recursos para calcular estos parámetros. Esta herramienta es especialmente útil en Montera, ya que permite que se almacenen únicamente cuatro elementos por *site* (media y desviación típica del tiempo de cola y los whetstones) en lugar de los datos relativos a la ejecución de miles de trabajos.

5.4. Caracterización de la infraestructura Grid

Es sabido que el rendimiento de una infraestructura Grid puede ser caracterizado con sólo dos parámetros (109; 110), el *asymptotic performance* y el *half performance length*.

El problema de esta propuesta es que está basada en el tiempo de ejecución de tareas de tamaño constante y este trabajo está centrado en variar el tamaño de las tareas. Por tanto, tuvo que ser modificada de manera que refleje el comportamiento de la infraestructura al ejecutar códigos de MC. A continuación se propondrá esa modificación junto a un algoritmo para calcular estos parámetros.

Como se ha explicado anteriormente, los códigos de MC han sido caracterizados empleando dos parámetros, *constant effort* y *sample effort*. Dependiendo del número de *samples* a ejecutar y los recursos disponibles, el esfuerzo computacional total necesario para ejecutar la tarea variará. Además, el tiempo de transmisión de los datos de entrada/salida y el tiempo de cola han de ser evaluados, debido a que representan una sobrecarga diferente dependiendo del tamaño de tarea. Aquí toda esta información es tenida en cuenta, como puede observarse en la Ecuación 5.2 que describe su cálculo.

$$\begin{aligned} Site\_performance \;=\; & (queue\_time + input\_data\_size/bandwith \\ & + output\_data\_size/bandwith \\ & + constant\_effort/whetstones \\ & + sample\_effort * num\_samples/whetstones) \\ & / num\_samples \end{aligned} \tag{5.2}$$

La unidad con la que se ha trabajado es *samples/segundo*, análogo al *tareas/segundo* del original (110).

Sin embargo, en el caso del algoritmo DyTSS esta caracterización presenta un inconveniente. Con la aproximación propuesta es necesario conocer la distribución exacta de tareas -esto es, el tamaño de cada tarea y en qué recurso será ejecutada- para poder calcular el rendimiento de la infraestructura. Por otro lado, al emplear el algoritmo DyTSS es necesario tener información acerca del rendimiento de la infraestructura para poder dividir la simulación en diferentes *chunks*. Para evitar este *deadlock* (interbloqueo) se emplea el algoritmo 2, descrito en la sección 4.4.2.

5.5. Planificación en dos niveles

Los planificadores actuales realizan la distribución de tareas en el recurso local -esto es, donde se están ejecutando- decidiendo el destino y tamaño de cada *chunk*. En Montera se sigue una nueva aproximación, permitiendo realizar la planificación tanto en el recurso local como en los remotos. El objetivo es permitir al *chunk* que modifique su tamaño dinámicamente, dependiendo del estatus del *site* remoto y las necesidades del usuario. Para llevar esto a cabo, Montera no sólo envía la tarea deseada, sino que además incluye toda la información disponible acerca del *site*, la aplicación a ejecutar y varios *scripts* a cargo de adaptar el tamaño del *chunk*.

5.6. Arquitectura

La figura 5.1 muestra la arquitectura de Montera desde un punto de vista de alto nivel. Como se puede apreciar, el *framework* está dividido en dos partes, Montera Local y Montera Remoto. Además hace uso del metaplanificador GridWay (35), tanto para obtener información como a la hora de la ejecución en Grid del código de Monte Carlo deseado.

A continuación se describirá la arquitectura de una manera más detallada.

5.6.1. Montera local

Montera Local representa el núcleo del *framework*. Su propósito es recibir las especificaciones del trabajo y requerimientos de el usuario, y ejecutarlo tan eficientemente como sea posible.

Para obtener la información acerca de los recursos Grid disponibles, el *Information Manager* emplea dos herramientas, GridWay y Montera Remoto. GridWay es empleado para obtener la información pública que los *sites* publican sobre si mismos, así como el número de tareas en ejecución en un momento cualquiera; Montera Remoto proporciona información acerca de la eficiencia de los *sites* después de la ejecución de cada tarea, además del

resultado del *benchmark* ejecutado -en caso de necesitarse- para calcular su rendimiento real.

Con esta información, el *Scheduler* decide el tamaño ideal y el número de *chunks* que ejecutar en cada *site*. Por último, el *Execution Manager* lleva a cabo estas ejecuciones.

A lo largo de este trabajo se han creado diferentes políticas de planificación. Para permitirlo, el *Scheduler* define una interfaz que debe ser implementada por cualquier política, con operaciones cubriendo la creación de *chunks* y su control. Para simplificar esta tarea -que puede ser realizada por cualquier usuario, no sólo por los desarrolladores de la aplicación- Montera incluye un toolbox con recursos útiles.

5.6.2. Montera remoto

La segunda parte del *framework* propuesto, Montera Remoto, optimiza la ejecución de la aplicación en los *sites* remotos. Es copiada a cada *site* junto a la aplicación a ejecutar, y decide el número exacto de *samples* que serán simulados en cada *chunk*.

Debido a la heterogeneidad de los *sites* remotos y las herramientas disponibles en cada uno, Montera Remoto ha sido implementado como un conjunto de *shell scripts*, con la intención de hacerlo tan portable como fuera posible. Su tamaño es muy reducido -del orden de decenas de kilobytes- por lo que la sobrecarga inducida por su copia a cada recurso es despreciable.

El primer *script*, *Main Script*, lee los datos de entrada, los requerimientos del usuario y la información almacenada sobre el rendimiento del *site*. Después llama a un segundo *script*, *Chosen Policy*, que calcula el número de *samples* a simular y lleva a cabo la ejecución. Por último, *Performance Analysis* realiza un análisis del rendimiento del *site* y devuelve esa información a Montera Local.

Montera Remoto incluye un conjunto de *Chosen Policy* para decidir el tamaño exacto de cada tarea. Cuando el usuario envía un trabajo a Montera elige cuál de ellas utilizar, y ésta es copiada al *site* remoto. Dependiendo de la funcionalidad deseada, el comportamiento de este *script* puede ser muy distinto: por ejemplo, *Deadline* calcula cuántos *samples* pueden ser ejecutados antes de un momento dado teniendo en cuenta la fecha de envío, el tiempo de cola y el ancho de banda; en *Flexible*, Montera Local proporciona un número máximo y mínimo de *samples* a simular y el *script* decide el número exacto dependiendo del rendimiento del *site* y el estatus actual. Es importante destacar que la combinación de diferentes políticas locales y remotas permite realizar planificaciones muy distintas, con lo que el usuario es capaz de optimizar la ejecución de su aplicación dependiendo de las necesidades del momento.

5.6.3. Interfaz de usuario

Al diseñar y construir Montera, se ha puesto un esfuerzo especial en mantener un API limpia y fácil de usar. De esta manera, si el usuario decide crear su propia política para Montera Remoto, sólo necesita crear un *script* con la funcionalidad deseada. Montera proporciona un conjunto de información que puede ser utilizada -tales como momento del envío y estatus del recurso- además de permitir el envío de cualquier información empleando la plantilla del trabajo.

Algorithm 3 Descripción del API de Montera Local.

```
public chunkList create_chunk_list (Application app, ResourceList ResourceList);
public int control_execution (Application app, ResourceList resourceList);
```

El algoritmo 3 describe el API proporcionado por Montera Local para crear una nueva política. Como se puede observar, basta con implementar dos funciones para definirla:

- `create_chunk_list` crea la lista de tareas que será enviada en primer lugar, al comienzo de la ejecución.
- `control_execution` controla la ejecución de las tareas: envía las que sean necesarias durante la ejecución, recibe los resultados parciales y decide cuándo se han ejecutado las suficientes.

Algorithm 4 Plantilla de Montera.

```
name=fafner.pbs
num_samples=500000
input_files=fafner.pbs, fafner_files.tar.gz
output_files=results${TASK_ID}.tar.gz
scheduling_policy=DyTSS
```

El algoritmo 4 muestra una plantilla típica de Montera, con los parámetros necesarios para una ejecución de FAFNER2:

- `name`: nombre de la aplicación a ejecutar.
- `num_samples`: número de *samples* deseados en la simulación completa.
- `input/output files`: ficheros de entrada y salida.
- `scheduling_policy`: política de planificación deseada.

Algorithm 5 Ejecución típica de Montera.

```
Export MONTERA_LOCATION=/path/to/Montera
java -Djava.library.path=/usr/local/gw/lib
-cp /usr/local/gw/lib/drmaa.jar:/usr/local/gw/include/:
/home/manuel/Montera_gw03/bin Montera template.mt
```

El algoritmo 5 muestra una ejecución típica de Montera. La llamada está dividida en tres secciones. La primera (`java`) arranca la máquina virtual; a continuación (`-Djava ... /bin`) se indican las librerías requeridas y sus localizaciones; por último (`Montera template.tm`) se señalan los parámetros de entrada a la máquina virtual de Java, que son la aplicación a ejecutar y su fichero de entrada.

Para facilitar la labor del usuario se ha creado un pequeño *shell script* que se encarga de encapsular los parámetros fijos, de manera que la ejecución se reduce a lo indicado por el algoritmo 6:

Algorithm 6 Ejecución típica de Montera (con *wrapper* que lo simplifica).

```
run_montera template.mt
```

Donde `template.mt` indica el template de Montera que se desea.

5.7. Limitaciones de la aproximación presentada

Como se ha explicado en la sección 3, es importante tener en cuenta que el concepto "Monte Carlo" engloba un amplio grupo de técnicas que, a pesar de guardar cierta relación entre si, difieren en diferentes aspectos.

En el caso de la aplicación y algoritmos aquí propuestos, la distribución dinámica de tareas presentada hace que sólo un subconjunto de los códigos Monte Carlo puedan ser ejecutados con Montera, aquellos cuyas iteraciones son independienes. Otro tipo de Monte Carlo presentan dependencias de datos entre las distinas iteraciones, con lo que su ejecución no puede ser dividida en fragmentos de tamaño arbitrario.

Quizá el ejemplo más claro lo constituyen los llamados *Random Walks* (70). Este algoritmo consiste en hacer avanzar al azar a un sujeto en un espacio de direciones previamente definido, de manera que cada paso consituye la simulación de un *sample*. En este caso, el resultado de cada *sample* (la posición del sujeto) no depende únicamente de la ejecución del mismo sino del estado previo a dicha simulación. Como se puede deducir fácilmente, la ejecución de un random walk no puede ser dividida en sub-tareas independientes sin que el resultado se vea directamente alterado.

Esta restrición hace que el estudio de algunas áreas del conocimiento quede fuera del alcance de Montera: en los problemas de plegado de pro-

teinas (115; 116; 117; 118), se emplean algoritmos basados en Monte Carlo replica sampling (119), Metropolis Monte Carlo (71) y random walks (70); en Lattice QCD (120) la posición de las partículas en el espacio-tiempo es determinada empleando Random Walks; en simulaciones de Lattice, se emplean Monte Carlos para generar una función de probabilidad(121); en química computacional el uso de Metropolis Monte Carlo está muy extendido para generar diferentes condiguraciones del sistema (122), aunque también son empleados algoritmos más sofisticados (123).

En cualquier caso, el efectuar un análisis en profundidad del uso en aplicaciones científicas de las diferentes técnicas basadas en Monte Carlo está fuera de la presente tesis doctoral. Si se desea más información al respecto, la bibliografía citada (especialmente el trabajo de Ferguson (123)) puede servir como una buena introducción al área.

Con esto, el objetivo de esta sección es únicamente servir para definir los límites de aplicación de la presente propuesta.

5.8. Conclusiones

En este apartado se ha presentado una aplicación de planificación llamada Montera.

Montera implementa mecanismos para recoger información relativa a los *sites* y la aplicación a ejecutar y proporciona al usuario la posibilidad de llevar a cabo un *scheduling* en dos niveles. Esta aproximación permite al usuario modificar la ejecución atendiendo a sus necesidades y a los requerimientos del experimento.

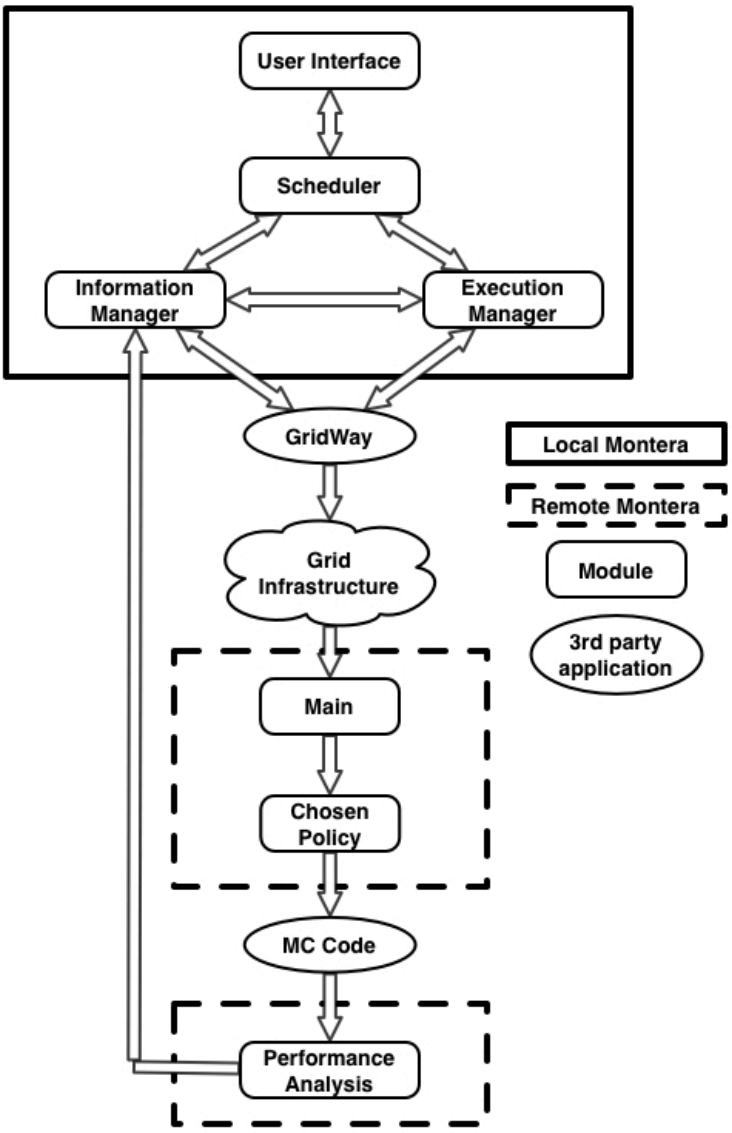

Figura 5.1: Descripción de alto nivel de la arquitectura de Montera.

Capítulo 6

Análisis de las propuestas

Tras haber expuesto las bases teóricas sobre las que se asienta la presente Tesis y detallado las innovaciones propuestas, es necesario analizar su rendimiento real. De este modo se comprobará que efectivamente el trabajo realizado es útil desde un punto de vista práctico y no una capa extra de software que añade complejidad al sistema de manera innecesaria.

6.1. Introducción

El análisis de rendimiento se llevará a cabo en dos fases.

En una primera se utilizará un simulador (MonteraSim) desarrollado para este propósito. De este modo se llevará a cabo un análisis preliminar del algoritmo de planificación propuesto, así como una comparación con los métodos estándar de planificación. Con ello, se podrá comprobar si, en condiciones ideales, el algoritmo DyTSS mejora las soluciones existentes. Además, el empleo del simulador permite el efectuar todas las pruebas y ajustes necesarios en un tiempo mínimo y sin emplear una infraestructura Grid real, evitando así su sobrecarga con tareas no dedicadas a la obtención de resultados útiles.

A continuación se llevará a cabo el análisis de rendimiento en un entorno real, las Organización Virtual *fusion* del proyecto EGEE/EGI y *prod.vo.eu-eela.eu* del proyecto GISELA. De este modo se podrá comparar Montera y el algoritmo DyTSS con las soluciones previas en una infraestructura en producción, de manera que los resultados que se obtengan sean reales y no estén alterados por el empleo de un entorno controlado.

Para llevar a cabo este estudio se han elegido las aplicaciones FAFNER2 (124), ISDEP (125) y FastDEP (126). Todas ellas son códigos de Monte Carlo empleadas en el CIEMAT para simular el comportamiento del reactor TJ-II, aunque pueden ser igualmente aplicados a las particularidades de otros dispositivos de tipo stellarator e incluso tokamaks. FAFNER2 es un simulador de inyección de partículas neutras en un reactor de fusión, mientras que ISDEP permite estimar el transporte colisional de iones. FastDEP es un *workflow*

que integra ambas, de manera que permite simular un mayor número de eventos del plasma.

Se ha optado por elegir dichas aplicaciones por su diferente comportamiento. FAFNER2 simula una gran cantidad de partículas -del orden de cientos de miles- con un tiempo de ejecución de varios segundos por partícula. En el caso de ISDEP, cada trayectoria simulada emplea cerca de una hora de CPU, necesitando decenas de miles de trayectorias por experimento. Así pues se consideró de interés el emplear ambas, y así poder comprobar si Montera es capaz de mejorar su rendimiento independientemente de estas características. El comportamiento de FastDEP es cercano al de ISDEP, ya que es esta aplicación la que consume la mayor parte del tiempo de computación empleado en la simulación.

6.2. Aplicaciones

Una información detallada de las aplicaciones puede encontrarse en los apéndices A, B y C, pero aquí serán brevemente descritas.

6.2.1. FAFNER2

La simulación por ordenador se ha posicionado como una de las herramientas más prometedoras para estudiar el comportamiento del plasma y la optimización de dispositivos de fusión antes de su construcción. Existen aplicaciones software que simulan diferentes partes y procesos de un reactor, obteniendo resultados muy precisos y que se corresponden exactamente con los fenómenos reales. De este modo, es posible estudiar el comportamiento de un reactor dentro de un laboratorio.

Uno de estos códigos es FAFNER2, que simula el calentamiento por NBI (*Neutral Beam Injection*, Inyección de Haces de Neutros), un dispositivo clave en el desarrollo de reactores de fusión. De hecho, esta técnica será empleada en el ITER (127), el reactor de fusión de nueva generación, donde FAFNER2 tendrá un rol relevante en su diseño y simulación.

El código original de FAFNER fue creado por G.C. Lister en el Max-Planck-Institüt en 1985 (124). La versión empleada en este trabajo es una adaptación de FAFNER para la geometría del stellarator TJ-II, que se encuentra en las instalaciones del CIEMAT. Fue desarrollada por el departamento de fusión del CIEMAT en colaboración con los autores originales (58).

La versión de FAFNER2 empleada (128) ha sido diseñada para su ejecución en plataformas distribuidas. Este proceso constituye parte del trabajo de tesis, y está detallado en el apéndice A.

6.2.1.1. La física de FAFNER2

El problema físico a resolver es el modelado de la inyección de haces de partículas neutras en plasmas tridimensionales con forma de toroide y el cálculo de la trayectoria de los iones rápidos resultantes hasta que se pierden o son absorbidos por el sistema.

La primera parte del código simula la inyección de partículas neutras en el toroide. Como resultado, se obtienen las coordenadas y velocidad de estas partículas.

A continuación, en la segunda parte del código, esta información es usada para determinar la ionización inicial de estos átomos en el plasma y después la distribución de energía en el plasma como resultado de su interacción con los iones rápidos. FAFNER2 también computa las pérdidas de energía que se producen cuando las partículas neutras colisionan con los conductos y aperturas. En este caso, el modelo de Monte Carlo se emplea para computar las coordenadas de un conjunto de iones rápidos, correspondientes al haz de neutros, así como los parámetros apropiados del plasma.

La trayectoria de estos iones y su interacción con el plasma son descritos en términos de una ecuación de Fokker-Planck (129). La trayectoria de los iones es resuelta empleando un sistema de coordenadas de Boozer (130). Estas técnicas numéricas son muy conocidas y han sido empleadas con anterioridad para describir el calentamiento mediante haces de neutros en dispositivos de fusión, tanto tokamaks como stellarators.

FAFNER2 proporciona dos conjuntos de datos tras cada ejecución. Primero se determina la posición y velocidad de un número determinado de partículas. Después, obtiene diferentes estadísticas acerca del estado del plasma: fracción de partículas ionizadas y perdidas, así como el total de energía que se ha suministrado al sistema.

6.2.2. ISDEP

ISDEP (*Integrator of Stochastic Differential Equations in Plasma*, Integrador de Ecuaciones Estocásticas Diferenciales en Plasma) (125) es un código que pretende resolver las ecuaciones del centro guía en presencia de colisiones con iones.

El plasma puede ser descrito con una ecuación de Fokker-Planck (129). Ésta es una ecuación extremadamente compleja e imposible de resolver matemáticamente, por lo que se emplea una equivalencia entre ella y las ecuaciones diferenciales de Langevin (131), centradas en una única particula en lugar de todo el plasma.

Esta equivalencia proporciona además una gran ventaja a la hora de resolver computacionalmente el problema: el obtener la información relativa a cada partícula en lugar de todo el plasma permite la creación de códigos paralelos o distribuidos, en que diferentes tareas calculan la información de

diferentes partículas, y posteriormente esta información es recopilada para obtener el resultado final. De este modo ISDEP ha sido ejecutado en diferentes plataformas distribuidas tales como BOINC (132) o Grid (133).

Para poder cumplir todos los requisitos impuestos por ISDEP, Montera fue empleado de una manera diferente a la del caso anterior. La razón de esto es que originalmente fue diseñada para códigos de MC que no necesitan manejar grandes cantidades de datos y con un ejecutable apropiado para copiarlo directamente al *site* remoto. En ISDEP esta aproximación no es válida, ya que además del ejecutable ha de copiar un archivo relativamente grande que contiene información acerca del dispositivo propuesto (su configuración magnética).

La solución propuesta pasó por emplear los *scripts* de *scheduling* de Montera Remoto de un modo distinto al planeado originalmente. Dichos *scripts* fueron planeados con la intención de ajustar el tamaño de los *chunks* en tiempo de ejecución (ver sección 5.6.2 para una descripción completa de los mismos), y aquí fueron modificados para que obtengan los datos de entrada de un *Storage Element*. De este modo se redujo en gran medida el tiempo de sobrecarga relacionado con la copia de los ficheros locales, además de permitir el emplear archivos que superan la limitación de tamaño del WMS. Como inconveniente, cabes destacar que el empleo de un *Storage Element* mediante funciones de gLite (*lcg-cp*) restringe su ejecución a *sites* que tienen instalado dicho middleware.

Por otro lado, GridWay -y por extensión, Montera- emplea Globus Toolkit (`gsiftp`) para transferir los ficheros de entrada a los *sites* remotos. Esta aproximación hace que no exista un tamaño máximo de fichero, pero obliga a copiar los ficheros a cada *site* remoto desde el recurso local.

Dependiendo de la configuración de los *sites* remotos, y teniendo en cuenta la información anterior, el usuario final ha de decidir cuál es la aproximación que más conviene a sus intereses: una mayor sobrecarga en la fase de envío (por el uso de `gsiftp`) o una limitación en el conjunto de *sites* que pueden ser empleados. Para maximizar los instrumentos que pueden ser empleados por el usuario final, en este trabajo se implementaron ambas aproximaciones.

6.2.3. FastDEP

FastDEP (nombre que proviene de FAFNER2 + ISDEP) consituye un *workflow* que pretende simular un mayor abanico de fenómenos del plasma.

Comprender la base física de FastDEP es inmediato una vez explicadas las dos aplicaciones que lo componen, FAFNER2 e ISDEP.

Como se ha descrito, el resultado de FAFNER2 está compuesto por un conjunto de partículas ionizadas en el plasma y una serie de datos estadísticos relativos a dichas partículas tales como velocidad o temperatura. Por otro

lado, la entrada de ISDEP consiste en una serie de partículas ionizadas en el plasma, de las cuales se desa conocer su trayectoria y evolución temporal.

Así pues la creación del *workflow* es inmediata: la salida de FAFNER2 es transformada para tener el formato deseado por ISDEP, que la utiliza como entrada. Este proceso es descrito en detalle en el apéndice C.

En este punto es posible proponer un avance: dado que ISDEP únicamente necesita información relativa a las partículas y no a la infrasestructura completa no es necesario el unir los datos de todas las tareas en las que se divide una ejecución de FAFNER2. Basta con que la salida de una de estas tareas sea la entrada de una tarea de ISDEP, consituyendo ambas una tarea mayor llamada FastDEP. De este modo los datos de entrada de FastDEP son los de la aplicación FAFNER2, y los de salida son los de ISDEP. A falta de un análisis en profundidad (descrito en el apéndice C) puede decirse que este proceso evita cuellos de botella, movimiento de dato entre distintos *sites* y tiempos de cola y planificación, con lo que se minimizan los *overheads* y aumenta el rendimiento.

6.3. Simulador

Montera incluye un simulador simple aunque práctico. Está acoplado al resto del código y emplea toda la información obtenida sobre la infraestructura en ejecuciones anteriores del programa para simular, de la manera más precisa posible, la ejecución de códigos de Monte Carlo en un entorno real.

Al disponer de información tanto de la aplicación como del rendimiento de los *sites*, este simulador puede calcular de una manera precisa el tiempo de ejecución de cada *chunk*, así como el tiempo esperado de cola y cuánto se empleará en la transmisión de los datos de entrada y saluda. Además, utiliza el número de recursos disponibles en ejecuciones previas para calcular el número y tamaño de los *chunks*.

Aunque el simulador obvia algunas de las características más importantes de la naturaleza de la Grid (dinamismo y alta tasa de fallos), es útil para proporcionar una estimación del rendimiento de cualquier política de planificación.

Las figuras 6.1 y 6.2 muestran la simulación de dos políticas diferentes, *EqualchunkSize* y DyTSS en el simulador.

Como puede verse, en estas simulaciones el empleo del algoritmo DyTSS no representa una mejora significativa respecto a una política de planificación clásica, basada en la partición de los *samples* a simular en *chunks* de tamaño idéntico. Esto es debido a la simplicidad de la simulación, llevada a cabo en un entorno controlado: no hay pérdidas de rendimiento, nuevos recursos o ningún otro factor, por lo que el algoritmo empleado no necesita adaptar dinámicamente el tamaño de los *chunks*. Como ha sido explicado, una de las ventajas de DyTSS respecto al resto de los algoritmos de *schedu-*

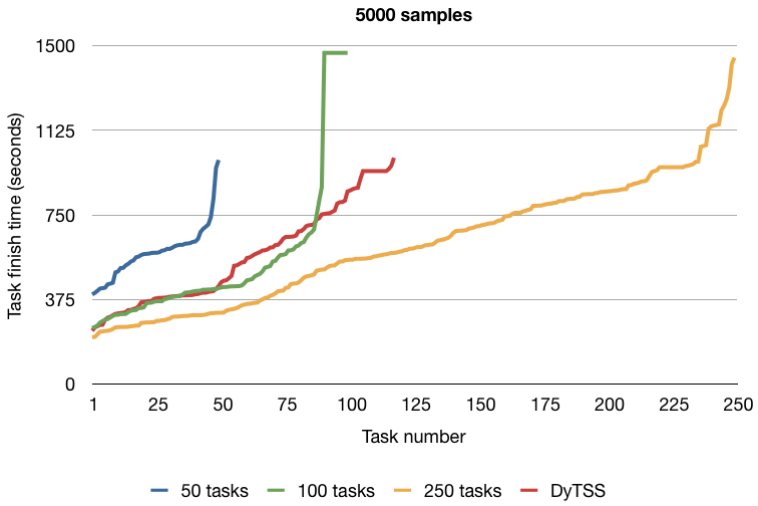

Figura 6.1: Simulación de una ejecución de FAFNER2 con 5.000 *samples*.

ling es su robustez, que no se hace patente en este caso. De todas formas, es importante destacar que, pese a todo, DyTSS nunca ofrece peores resultados que los demás. Por tanto, su utilización en ningún caso supone una pérdida de rendimiento.

Por otro lado, con la política *EqualchunkSize* los resultados varían dependiendo del tamaño de los *chunks* y el número de *slots* libres. En esta simulación se emplearon 90 *slots*, que corresponde a los recursos que estaban habitualmente a disposición del usuario empleando el banco de pruebas descrito en la sección 6.4.1. En el primer caso, las 50 tareas empiezan su ejecución inmediatamente, así que no hay *overhead* debido al tiempo de espera. En el caso de las 250 tareas, hay un tiempo de cola significativo en dos tercios de las tareas, pero el que haya un gran número de éstas permite el realizar una planificación eficiente, mandando más tareas a los recursos más rápidos. Por último, en el caso de 100 tareas, hay 10 con un tiempo de cola significativo, pero no hay suficientes como para poder realizar ningún tipo de *scheduling*. Este hecho se ve reflejado en la pendiente de las gráficas: un crecimiento lento o nulo significa que hay tareas acabando a un ritmo alto, y un crecimiento rápido o infinito significa que las tareas están en ejecución o esperando. En el caso de DyTSS la gráfica crece con un valor constante, lo que significa que las tareas son ejecutadas progresivamente y emplean el menor tiempo de ejecución posible. En otras palabras, en el caso de DyTSS el número y tamaño de las tareas ha sido adaptado a la infraestructura Grid, por lo que su ejecución es siempre eficiente.

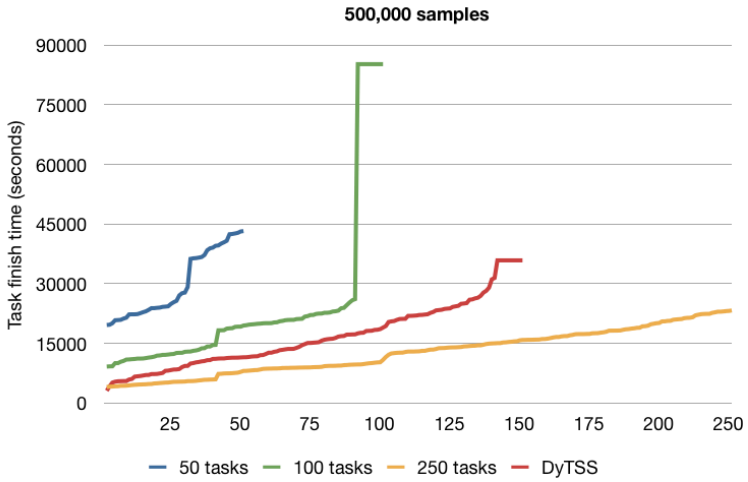

Figura 6.2: Simulación de una ejecución de FAFNER2 con 500.000 *samples*.

6.4. Experimentos y resultados

Tras llevar a cabo las simulaciones de DyTSS en un entorno controlado, se llevaron a cabo diferentes experimentos para comprobar la eficiencia de Montera en un entorno Grid real.

Para ello, se realizaron dos experimentos diferentes: Política *Deadline*, donde todos los recursos disponibles son empleados para simular tantos *samples* como sea posible antes de un *deadline* determinado, y Política DyTSS, donde el algoritmo propuesto es comparado con otros para demostrar su superioridad.

6.4.1. Banco de pruebas

El análisis de rendimiento de Montera fue llevado a cabo en la VO *fusion* de la infraestructura Grid EGI (31). En el momento en que estas pruebas se llevaron a cabo (Julio 2010), esta VO estaba compuesta por 45 nodos con más de 15.000 CPU, que proporcionaron un tiempo total de computación de más de 2500 KSI2K (Kilo SpecInt2000 (134)) horas durante el 2009 sólo en *sites* europeos. Por supuesto, no todas estas CPU pueden ser utilizadas por un sólo usuario al mismo tiempo, sino que existen políticas de uso muy estrictas para evitar abusos.

Como se ha mencionado anteriormente, el código de Monte Carlo elegido para llevar a cabo el experimento es la versión Grid de FAFNER2. Por el diseño del código, una instancia de FAFNER2 puede calcular hasta 8.000 *samples*, lo que establece un límite superior al tamaño de cada *chunk*.

El recurso local donde Montera fue ejecutado consistió en una máqui-

na virtual corriendo Debian 4 con GridWay 5.4.0 y Java Virtual Machine 1.5.0.09. Este recurso estaba en estado de producción -siendo empleado por distintos usuarios para realizar su trabajo diario- con una configuración bastante restrictiva: un intervalo de *scheduling* de 30 segundos, una tasa de envio de 15 tareas/ciclo, y un número máximo de trabajos por usuario de 100.

De hecho, estas restricciones limitan la escalabilidad de la solución propuesta, añaden *overhead* y evitan que se pueda obtener un mayor grado de paralelismo. La decisión de no modificarlas, ajustando la configuración de GridWay para obtener el mejor resultado posible, es parte de la determinación de crear una aplicación y algoritmo de *scheduling* que supere a los ya existentes en entornos de producción y en condiciones de trabajo reales. Esta configuración particular fue establecida por el administrador del recurso local, siendo la instancia de GridWay utilizada por varios usuarios cuando esta prueba estaba siendo realizada. Por tanto, la posibilidad de ajustar el sistema fue completamente descartada.

Hay que señalar que para la ejecución de FastDEP también se incluyó un servidor web en el recurso local antes descrito. Como se indica en el apéndice C, este servidor es necesario para establecer contacto con el usuario de la aplicación, de manera que la toma de decisiones sobre la ejecución del mismo sea más ágil y cómoda que empleando una terminal de texto. Dicho servidor consistió en una instalación *LAMP* (135) (Linux-Apache-MySQL-PHP) estándar, con Apache 2.2.17, MySQL 5.0.77 y PHP 5.3.6.

6.4.2. Resultados obtenidos

Después de realizar las simulaciones previas de DyTSS en un entorno controlado, se ejecutaron diferentes pruebas para comprobar la utilidad y correcto funcionamiento de Montera en un entorno Grid real.

Para ello se llevaron a cabo una serie de experimentos, centrados en los diferentes aspectos de Montera y el algoritmo DyTSS.

El primero es la comprobación práctica de la adaptación de Montera al número de *slots* disponibles en un *site*, descrita de manera teórica en el la sección 4.4.2. De este modo se verifica el correcto funcionamiento de Montera y del algoritmo DyTSS en la obtención de información sobre la infraestructura, necesario para llevar a cabo el resto de experimentos.

A continuación se muestra el resultado de una ejecución de Montera con política de planificación *Deadline* (mencionada en la sección 5.6.2), comprobando así la eficiencia de la planificación en dos niveles y la precisión en la caracterización de la infraestructura y la aplicación. Tanto en este caso como en el anterior se empleó el código FAFNER2.

Por último se efectuó un profundo análisis del algoritmo DyTSS. En este caso no sólo se empleó FAFNER2 sino que se incluyeron ISDEP y FastDEP en el análisis, para determinar la eficiencia del algoritmo con distintos có-

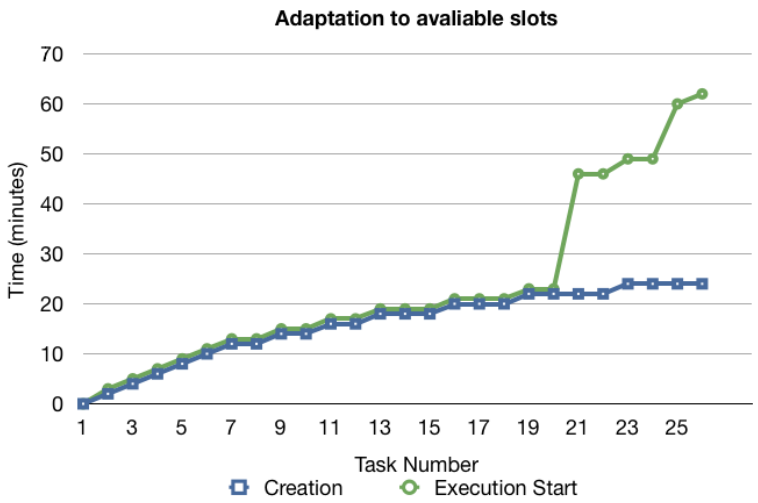

Figura 6.3: Adaptación al número de *slots* disponibles en un *site* con 20 *slots* libres.

digos Monte Carlo: FAFNER2 como ejemplo de aplicación con multitud de simulaciones extremadamente rápidas (del orden de un segundo de CPU por partícula), ISDEP como aplicación con simulaciones más costosas computacionalmente (una hora por partícula) y FastDEP siendo un híbrido de ambas. De este modo, si Montera con DyTSS consigue reducir el tiempo de ejecución de las tres, podrá cocluirse que la propuesta es válida para la generalidad de los códigos Monte Carlo.

6.4.2.1. Adaptación dinámica al número de *slots* libres con la política de *scheduling* DyTSS

Como ha sido explicado, empleando DyTSS Montera es capaz de detectar el número de *slots* disponibles en cada recurso y adaptar dinámicamente la carga de trabajo en consecuencia. En la figura 6.3 se presenta gráficamente una prueba de su funcionalidad.

Este experimento consistió en la ejecución de DyTSS en un único *site* Grid, `ce01-tic.ciemat.es`. Después del *benchmarking* del *site*, la ejecución empieza con una sola tarea. Montera envía un 20 % más de tareas (que en este caso es una pues el rendondeo siempre es al número mayor), y como hay *slots* libres, ésta comienza su ejecución, con lo que el margen del 20 % desaparece. Cuando Montera lo detecta crea otra tarea para cumplir el 20 % adicional de margen de nuevo. Este proceso es repetido hasta que hay 20 tareas ejecutándose al mismo tiempo, alcanzando el número de *slots* libres que pueden ser utilizados por un solo usuario en el *site*. Después de ese momento, las tareas sólo son creadas y ejecutadas después de la finalización

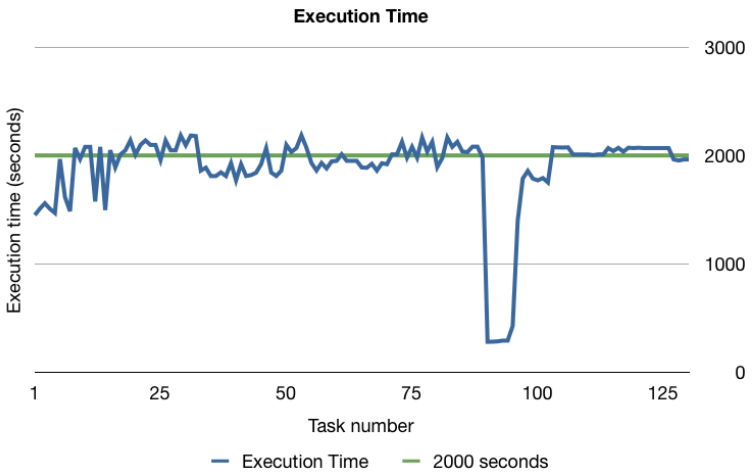

Figura 6.4: Política *Deadline*.

de las anteriores.

Cuando esta simulación ha terminado, la información acerca del número de *slots* libres es almacenada, y en el futuro será empleada para determinar el número de tareas que crear al comienzo de la ejecución.

6.4.2.2. Política *Deadline*

Como se ha explicado, es posible emplear distintas políticas de planificación con Montera. En este caso se propone la política *Deadline*, donde Montera envía una tarea a cada *slot* libre y ahí ejecuta tantas simulaciones como sea posible antes de un momento determinado.

Esta es una política interesante porque ofrece una nueva posibilidad que -hasta donde alcanza el conocimiento del autor- era imposible de llevar a cabo eficientemente con los algoritmos y herramientas de *scheduling* previamente existentes.

Esta política está dividida en dos secciones, una en Montera Local y otra en Montera Remoto.

En Montera Local se ha creado una nueva política mediante la implementación de la interfaz mostrada en el algoritmo 3 de la sección 5. Se crea una nueva tarea para cada *slot* libre, incluyendo el momento de creación de la tarea y el tiempo disponible para la ejecución como variables de entorno del *site* remoto. En Montera Remoto, el *script* que determina el número de *samples* a simular lee esta variable de entorno, la hora local, el rendimiento del *site* y el *profiling* de la aplicación, para a continuación calcular el número de *samples* que pueden ser simulados en el tiempo disponible. De este modo el tiempo de cola, migración de tareas o cualquier otro evento relacionado

con la ejecución en Grid no tiene influencia en el cumplimiento del plazo deseado: si una tarea empieza su ejecución instantáneamente simulará un número determinado de tareas, y si comienza más tarde dicho número de tareas será menor.

Los resultados de esta política pueden verse en la figura 6.4, donde la finalización de las tareas está representada en azul y el tiempo deseado de finalización (2.000 segundos) en verde. En este caso sólo se ha llevado a cabo una ejecución, para así poder poner resultados realistas. Se consideró que el repetir el experimento sucesivas veces y utilizar estadística para unir los resultados eliminaría las variaciones producidas por las imprecisiones del método empleado y factores exógenos a Montera (tales como variaciones en el rendimiento de los procesadores de los *sites*) y no reflejaría una ejecución real de Montera con la política *Deadline*.

En este experimento el resultado obtenido es positivo. De las 134 tareas enviadas únicamente fallaron 7 (un 5.22 %) y el resto empleó un tiempo medio de ejecución de $(195 \pm 16) \cdot 10$ segundos, lo que supone un error del 2.1 % y una desviación típica del 8 %. En cualquier caso esto puede ser solucionado, por ejemplo, fijando un *deadline* estricto y cancelando las que no hayan acabado después de un margen de tiempo. Montera proporciona un amplio conjunto de herramientas a disposición del usuario, por lo que estas modificaciones pueden ser llevadas a cabo sin demasiado esfuerzo.

6.4.3. Política de planificación DyTSS

Por último, se llevó a cabo un análisis de la política de planificación DyTSS. Para ello se emplearon tres aplicaciones con diferentes comportamiento, de manera que las ventajas obtenidas con dicho algoritmo pudieran extraporlarse a la generalidad de los códigos Monte Carlo y no a una aplicación concreta.

6.4.3.1. Política de planificación DyTSS con FAFNER2

En este apartado se demuestra la eficacia del algoritmo de planificación DyTSS frente a dos políticas distintas, una distribución *Equal Size* y GTSS. Estas políticas han sido elegidas por ser especialmente representativas: *Equal Size* puede considerarse una estándar ampliamente empleada en este tipo de problemas y GTSS ha demostrado -como se explicó en la sección correspondiente- ser el algoritmo de *self-scheduling* más eficiente para infraestructuras Grid controladas.

Como en el caso de las simulaciones, se han realizado tres experimentos con un tamaño de tarea creciente. El número de *samples* ha sido 5.000, 25.000, 50.000, 200.000 y 400.000 (entre 4 y 180 horas de CPU), lo que pueden considerarse simulaciones cortas, medias y largas acorde al tipo de simulaciones para el que FAFNER2 es empleado.

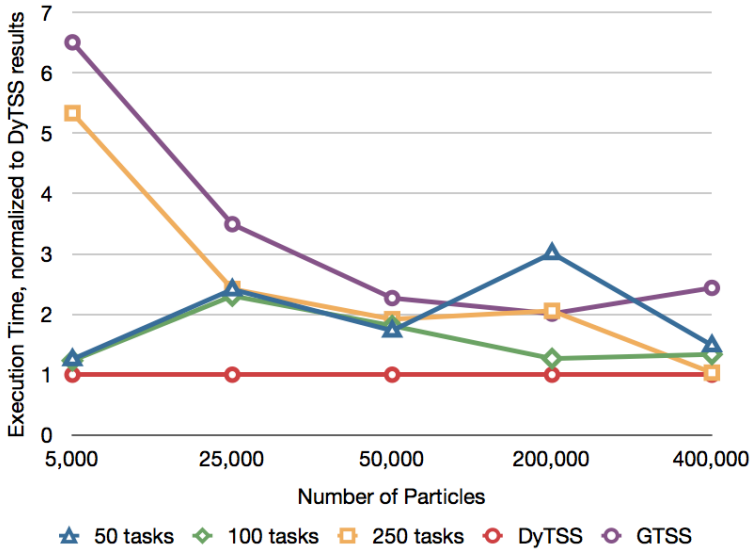

Figura 6.5: Ejecución de FAFNER2 con diferentes políticas y un número creciente de partículas.

En el caso de las pruebas de *Equal Size*, se han realizado algunos ajustes para maximizar su rendimiento: se ha fijado un umbral de *re-scheduling* de 300 segundos y un tiempo de cola máximo de 600 segundos, con el fin evitar los recursos saturados y empezar la ejecución tan pronto como fuera posible. Además, se ha empleado la política de planificación por defecto de GridWay -cuya eficiencia ha sido previamente demostrada en (9)- para elegir el *site* donde ejecutar cada tarea. La razón para llevar a cabo estas optimizaciones es asegurarse que cualquier mejora obtenida con Montera y DyTSS proviene de la aplicación en sí, no de factores exógenos como el empleo de GridWay.

Además, es necesario hacer algunas puntualizaciones sobre las pruebas del algoritmo GTSS. Como se ha señalado previamente, la información acerca de la estructura tiene que ser obtenida antes de la ejecución de cualquier algoritmo de *self-scheduling*. En este caso, esta información ha sido obtenida con Montera empleando los instrumentos antes descritos. La alternativa usar información pública acerca del *site*- lleva a resultados incorrectos y problemas en la ejecución. Por ejemplo, en el *site* Grid del CIEMAT (`ce01-tic.ciemat.es`) se anuncia la existencia del 208 *slots* libres, pero un único usuario no puede utilizar más de 20 al mismo tiempo. Si se hubiera empleado esta información con el algoritmo GTSS el número de *chunks* enviados a la Grid sería extremadamente grande, afectando negativamente al rendimiento.

La figura 6.5 muestra los resultados de la ejecución de FAFNER2 con distintos tamaños de problema y diferentes políticas de planificación, las mismas

que se emplearon en el simulador. Aquí los resultado están normalizados al tiempo de ejecución del algoritmo DyTSS, lo que permite apreciar de manera inmediata las diferencias de rendimiento entre los distintos algoritmos.

El realizar simulaciones empleando 5.000 a 400.000 partículas viene dado por el problema a resolver y la arquitectura de FAFNER2. El límite inferior, 5.000 partículas, corresponde a la mitad de un caso de uso pequeño, que suele ser de aproximadamente 10.000. El límite superior, 400.000, es el producto de 8.000 -máximo número de partículas que, por diseño, puede simular una instancia de FAFNER2- por 50 tareas, que es la política elegida con menos tareas.

En el caso de *Equal Size*, los resultados son consistentes con los obtenidos con el simulador. Con un número pequeño de partículas, la ejecución en Grid tiende a beneficiar a un número pequeño de tareas, dado que los *overheads* son significativos y representan una alta proporción del tiempo de ejecución. Sin embargo, cuando el número de tareas crece el *overhead* no es tan importante. En ese caso un *scheduling* adecuado es de vital importancia, y es mejor ejecutado con un número mayor de tareas que distribuir. Esto se aprecia en la gráfica comparando los resultados para 50 y 250 tareas. En el primer caso la tendencia de la gráfica es claramente descendente, mientras que en el segundo es en general creciente.

Los resultados obtenidos con GTSS no fueron tan buenos en el entorno real como en las simulaciones. Como se explicó previamente, GTSS crea una relación fija entre tareas y recursos al comienzo de la ejecución. Por tanto, el fallo o pérdida de rendimiento de un solo *site* puede ralentizar toda la ejecución y aumentar el *makespan*. A pesar de haber empleado la información obtenida con Montera para minimizar este problema, el resultado de emplear el algoritmo en infraestructuras dinámicas resultó ser inferior.

Por último, DyTSS obtuvo el mejor tiempo de ejecución en todos los casos. Con una mejora del 650 % en el mejor de los casos y del 3 % en el peor, su distribución dinámica de tareas y adaptación a recursos inestables resultó ser la mejor alternativa de entre las elegidas. Esto es importante porque permite una ejecución desatendida del usuario, y aunque este no sea experto se garantiza que el rendimiento obtenido es el mayor posible de entre las alternativas conocidas.

La tabla 6.1 muestra los tiempos de ejecución y desviación estándar para todos los experimentos llevados a cabo con FAFNER2. Como puede verse, DyTSS siempre proporciona los mejores tiempos de ejecución, y la mayoría de las veces es el que tiene una menor desviación. La dinamicidad de las infraestructuras Grid y su variedad de recursos provocan que el tiempo de ejecución de las aplicaciones sea siempre diferente, pero esta desviación estándar puede dar una estimación de cómo funcionan los algoritmos: una menor desviación significa que el algoritmo de *scheduling* es capaz de sobreponerse a pequeñas diferencias en la infraestructura entre diferentes ejecuciones,

Samples	*50 tareas (segundos)*	*100 tareas (segundos)*	*250 tareas (segundos)*	*GTSS (seconds)*	*DyTSS (seconds)*
5,000	1331 ± 85,56 %	1307 ± 55,77 %	5649 ± 65,96 %	6892 ± 49,11 %	1059 ± 48,91 %
25,000	4768 ± 67,39 %	3653 ± 60,75 %	6361 ± 69,23 %	8038 ± 50,03 %	2663 ± 48,16 %
50,000	5857 ± 61,36 %	6131 ± 44,07 %	6496 ± 9,98 %	7681 ± 15,45 %	3385 ± 23,84 %
200,000	15665 ± 12,14 %	6166 ± 1,04 %	8758 ± 21,51 %	10456 ± 5,56 %	5200 ± 11,57 %
400,000	16647 ± 15,17 %	14963 ± 25,03 %	11568 ± 10,63 %	27284 ± 21,98 %	11192 ± 12,03 %

Tabla 6.1: Tiempo de ejecución y desviación estándar de los valores mostrados en la Fig. 6.5.

y una gran desviación implica que el tiempo de ejecución es ampliamente dependiente del sitio que ejecuta cada tarea.

Esta tabla muestra así mismo que la desviación típica es normalmente mayor cuando el número de tareas en el que se ha dividido la aplicación es menor: el número de *samples* a simular en cada *site* es mayor y por tanto también lo es el esfuerzo computacional, por lo que el rendimiento del *site* que ejecuta esa tarea se torna más importante. Así mismo, un número más pequeño de tareas reduce las posibilidades de *scheduling* y la obtención de un tiempo de ejecución satisfactoria pasa a depender del azar.

6.4.3.2. Política de planificación DyTSS con ISDEP

Como se ha descrito, ISDEP calcula la solución a las ecuaciones del centro guía en la presencia de colisiones con iones. En el caso de uso empleado en este trabajo la división del problema ha sido hecha en trabajos que integran diez trayectorias del reactor de fusión LHD(136), esto es, un trabajo calcula ese número de tareas. La ejecución de uno de estos trabajos toma aproximadamente ocho horas en un equipo de escritorio (PC, Pentium IV@3GHz, 2 GB RAM).

Para llevar a cabo un experimento útil desde el punto de vista físico, se necesitan un mínimo de 50.000 trayectorias, lo que -como el lector comprenderá- necesita una enorme potencia computacional. En la presente sección se presentan resultados correspondientes a 5.000 trayectorias; se ha elegido ese número ya que permite mostrar las ventajas de utilizar Montera para ejecutar ISDEP sin la necesidad de llevar a cabo un experimento completo, cuyos resultados y análisis de los mismos están fuera del ámbito de este trabajo.

parámetro	*250 tareas*	*500 tareas*	*GTSS*	*DyTSS*
Execution time [s]	194828	222061	244661	183956
Makespan normalizad to DyTSS	1,05	1,21	1,33	1
Standard Deviation [%]	39,45	32,30	29,68	7,40

Tabla 6.2: Resultados de la simulación de 5.000 partículas con ISDEP con distintas políticas de planificación.

El funcionamiento de un experimento completo es idéntico al de una sección del mismo: se envía el número deseado de trabajos idénticos, que solamente difieren en una semilla aleatoria. Así pues, los resultados aquí expuestos permiten decidir acerca de la utilidad de Montera para emplear ISDEP.

Es importante señalar que la agrupación de partículas a simular en diferentes tareas es diferente en el caso de ISDEP y de FAFNER2. Originalmente se planeó el llevar a cabo la misma -esto es, dividir las 5.000 simulaciones de ISDEP en 50, 100 y 250 tareas- pero surgió un problema inesperado. La mayoría de los sitios Grid tienen limitada la duración máxima de los trabajos a un periodo que oscila entre las 24 y las 48 horas, por lo que la simulación de más de 20 partículas en una sola ejecución de ISDEP era habitualmente cancelada. Con esta limitación en mente se optó por dividir el problema en 250 y 500 tareas que simulan 20 y 10 partículas cada una respectivamente.

La tabla 6.2 muestra el tiempo de ejecución de ISDEP en la VO *fusion*, de manera análoga a 6.1.

Como puede observarse, el porcentaje de mejora obtenido empleando DyTSS oscila entre un 5 % y un 33 % comparado con la mejor y peor de las alternativas. Así mismo puede observarse que DyTSS es el algoritmo de planificación en el que la desviación estándar de los resultados es menor, lo que apunta directamente a una mayor robustez y adaptabilidad frente a cambios externos.

A pesar de no ser espectacular este resultado es enormemente interesante, especialmente si es tenido en consideración junto al caso anterior: con FAFNER se ha simulado un altísimo número de partículas con un corto tiempo de ejecución por cada una, mientras que en ISDEP se ha utilizado el enfoque contrario, simulando un número más reducido de partículas en las que cada una tiene un tiempo de ejecución elevado. Al demostrar empíricamente que Montera es capaz de proporcionar un rendimiento adecuado en ambos casos, puede deducirse que la aproximación aquí propuesta es válida sea cual sea el tiempo de simulación de cada muestra.

Es importante destacar que en el caso de la VO empleada todos los *sites*

Scheduler	Walltime [dd:hh:mm]	Eficiencia [%]	Tasa de fallos [%]
Euler	2:00:23	100 %	0 %
Montera	2:12:26	80,06 %	-
GridWay	2:22:15	68,89 %	0 %
WMS	92:03:43	52,57 %	11,80 %

Tabla 6.3: Tiempo de ejecución y tasa de errores de FastDEP con distintos *schedulers*.

han de soportar gLite y las instrucciones de acceso a *Storage Elements*, por lo que cualquiera de las anternativas expuestas en el apéndice B.2 (empleo de `lcg-cp` o copia de los datos de entrada desde el equipo local) son válidos y permiten emplear el mismo abanico de recursos. A la hora de descargar los datos de entrada desde un *Storage Element* a los *worker nodes* donde se ejecutó la aplicación, diferentes pruebas dieron como resultado un tiempo medio de transferencia del archivo de entrada de aproximadamente 8 segundos -esto es, un ancho de banda de bajada de aproximadamente 1 MB/s. Dado que el tiempo de ejecución de cada tarea fue de varias horas, se consideró que el *overhead* producido al copiar los datos desde el sitio local en lugar de emplear un *Storage Element* era aceptable y compensaba el mantener una arquitectura del sistema lo más simple posible.

6.4.3.3. Política de planificación DyTSS con FastDEP

Como se ha descrito previamente, FastDEP calcula la inyección de partículas neutras en el plasma y su evolución temporal en términos de posición y energía.

Para llevar a cabo el presente análisis del rendimiento, se ha ejecutado un experimento consistente en la ejecución de 30.000 trayectorias independientes. Esta es una muestra lo bastante significativa como para extrapolar los resultados a experimentos mayores y, a la vez, su reducido tamaño permite el repetir el experimento varias veces y analizar las diferencias estadísticamente. Los experimentos más largos -es habitual emplear FastDEP para simular varios cientos de miles de trayectorias- se llevan a cabo ejecutando tareas idénticas que únicamente difieren en una semilla aleatoria.

En el análisis de rendimiento de Montera con FastDEP se ha seguido una aproximación ligeramente disinta a la de los ejemplos anteriores. En este caso se ha optado por efectuar una división del problema en tareas del mismo tamaño y ejecutar la aplicación en distintas plataformas, comparando estos resultados a los obtenidos empleando Montera. El proposito de esto es poder comparar el rendimiento de las distintas plataformas de una manera

Scheduler	**Tiempo Total [dd:hh:mm]**	**Ejecución [dd:hh:mm]**	**Cola [dd:hh:mm]**	**Overhead [%]**
Euler	523:18:26	244:21:47	278:20:39	53,24 %
Montera	-	278:09:07	-	-
GridWay	709:22:53	325:03:08	324:01:04	45,64 %
WMS	1260:07:58	385:21:48	874:10:10	69,38 %

Tabla 6.4: Tiempo de ejecución acumulado ejecutando FastDEP con distintas plataformas.

sencilla y ver así qué parte del aumento de rendimiento obtenido corresponde a Montera, qué parte a GridWay y qué parte al hardware subyacente.

La tabla 6.3 muestra el tiempo de ejecución al ejecutar FastDEP en diferentes infraestructuras con 4 herramientas diferentes: *Euler*, WMS, una instalación estándar de GridWay y el *framework* Montera. Junto a la tabla 6.4, puede determinarse el comportamiento de las soluciones propuestas.

El primer hecho destacable de estos resultados es que *Euler* obtiene el mejor tiempo de ejecución tanto en términos de *walltime* como de *overhead*. El hecho de que todas las plataformas tengan el mismo límite superior de recursos a emplear es una ventaja significativa para *Euler*, ya que está formado por hardware homogéneo y actualizado. Además, ninguno de los *overheads* inherentes a las infraestructuras Grid se aplica aquí. Combinando esto con la mayor estabilidad de los cluster locales frente a los recursos remotos, el resultado es una plataforma rápida y sólida donde ejecutar las aplicaciones.

De entre las soluciones Grid propuestas, una vez más Montera sobresale frente al resto y GridWay supera ampliamente a WMS. La eficiencia de Montera es de un 80,06 % frente a la obtenida con *Euler*, mientras que en el caso de GridWay y WMS esta eficiencia cae al 68,69 % y 52,57 % respectivamente.

La tabla 6.4 saca a la luz ciertos hechos interesantes acerca del tiempo de ejecución en las diferentes plataformas. Por ejemplo, no hay una relación directa entre el *overhead* y la eficiencia de cada plataforma, dado que es calculado como un porcentaje del tiempo final. el caso de *Euler* es el más llamativo: tiene un mayor *overhead* que GridWay, pero éste viene dado por su reducido tiempo de ejecución. Además, en el caso de Montera no hay *overhead* que cuantificar como tal: este *framework* está continuamente creando y cancelando tareas debido a varias razonas (tiempo de cola, nuevo recurso descubierto o modificación de uno existente, etc.). Estas tareas tienen una función clave en la obtención de información y toma de decisiones de los algoritmos de planificación, por lo que no deben ser tomados como *overhead* o ejecuciones fallidas. En este caso no tiene sentido medir su tiempo de cola, y proporcionar un *overhead* basado únicamente en el tiempo de transmisión de los archivos de entrada distorsionaría cualquier comparación con el resto.

La inclusión de GridWay en este análisis del rendimiento no pretende representar una comparación a fondo entre GridWay y WMS, dado que esta ya ha sido llevada a cabo en estudios como (38). Al contrario, este análisis pretende enfocarse como una comparación de las herramientas más utilizadas en el "mundo real". WMS es el *scheduler* oficial en muchos proyectos Grid dada su inclusión en gLite; GridWay se ha planteado como una potente alternativa a WMS y adquirido una significativa base de usuarios, principalmente debido a su simplicidad, robustez y un rendimiento superior. Montera intenta ocupar el espacio actualmente vacío de un *scheduler* dedicado a los códigos Monte Carlo, por lo que debería superar en rendimiento a ambos. Además, la inclusión de GridWay permite poder estimar la proporción de mejora frente a WMS debida a GridWay y debida a Montera, para poder así demostrar la viabilidad de nuestra propuesta.

Por último es necesario tomar ciertas consideraciones acerca de la tasa de fallos mostrada en la tabla 6.3: WMS tuvo una tasa de fallos del 11,80 %, mientas que en el caso de los dos planificadores basados en GridWay fue de cero. Al emplear WMS el usuario tiene que estar al tanto del resultado de la ejecución y cualquier error que pueda surgir para solucionarlo manualmente -por ejemplo, enviando la tarea de nuevo. GridWay incorpora un mecanismo de manejo de errores, por lo que cualquier tarea fallida es enviada automáticamente de nuevo hasta que finaliza exitosamente. Montera no tiene que enfrentarse a este problema ya que, como ha sido previamente explicado, no necesita que el resultado de cada tarea sea exitoso para que sí lo sea el resultado global de la ejecución.

Este resultado es consistente con lo obtenido en los experimentos referenciados en la bibliografía, mostrando una vez más la robustez de GridWay frente a WMS como planificador sobre el que desplegar nuestras aplicaciones.

6.4.4. Sobrecarga

Como en cada experimento que supone la modificación de la arquitectura software o hardware de una infraestructura, es necesario medir la sobrecarga inducida. En este caso hay dos etapas cuya mejora de eficiencia es necesario probar: la sustitución de WMS por GridWay, y la adición de Montera.

Como se ha explicado en el caso de uso relativo a FAFNER2, GridWay ha sido configurado con un conjunto de parámetros muy restrictivo, de manera que se limita la carga del sistema y éste se mantiene todo lo estable posible, dado que se trata de un entorno de producción. Debido a eso, aparece cierta sobrecarga que es necesario analizar.

El establecimiento de un intervalo de planificación de 30 segundos añade una sobrecarga media de 15 segundos por tarea. De todos modos, dado que el tiempo total de ejecución de estos experimentos fue del orden de miles de horas de computación, se puede deducir que esta sobrecarga no influyó

significativamente en el tiempo total de ejecución. Lo mismo ocurre con el máximo de 15 tareas enviadas por intervalo de ejecución: aunque puede representar un cuello de botella en aplicaciones compuestas por un conjunto enorme de tareas muy cortas, en los casos que nos ocupan su influencia es despreciable.

Montera ha sido implementado en Java y añade una carga muy pequeña al sistema: su tiempo de ejecución es de varios segundos. Más allá de la necesidad de una *Java Virtual Machine* (JVM), Máquina Virtual de Java, sus requisitos de CPU y memoria son despreciables.

La sobrecarga más importante que Montera provoca en el sistema se dan en las fases de *profiling* de los recursos y la aplicación: se envía una tarea de trescientos (300) segundos cada vez que se descubre un nuevo recurso y, para llevar a cabo el modelado de la aplicación, se ejecutaron en paralelo diez tareas con un tiempo de ejecución de unos diez mil (10.000) segundos (unas 2.7 horas) cada una.

Dependiendo del uso de Montera, la influencia de estos mecanismos es muy variable. Montera ha sido diseñado para un uso continuado, por lo que la primera vez que es ejecutado emplea un largo periodo de tiempo analizando la infraestructura y la aplicación. Por tanto, si la intención del usuario es llevar a cabo sólo una ejecución de una aplicación concreta, y GridWay no está disponible, entonces debería utilizar WMS. Si el usuario tiene la posibilidad de utilizar GridWay, ésta es la mejor alternativa para pequeños experimentos, debido a la mejora de rendimiento que supone. Y si la intención del usuario es emplear la Grid de una manera regular y prolongada, el aumento de rendimiento que se obtiene con Montera compensa claramente su largo periodo de inicialización. Además, el hecho de que tanto el análisis de la infraestructura como el de la aplicación son llevados a cabo de manera desatendida ayuda a enmascarar esta sobrecarga, permitiendo al usuario actuar según el paradigma "envía y olvídate".

En el caso de FastDEP hay una sobrecarga adicional asociada a la aplicación auxiliar que adapta la salida de FAFNER2 para convertirla en la entrada de ISDEP. Esta aplicación lee un fichero de texto con una logitud máxima de 8.000 líneas, extrae información de cada una de ellas y la vuelca a un fichero binario de datos. Dado que el único procesamiento necesario es parsear el fichero de texto, esto se lleva a cabo muy rápidamente y su tiempo de ejecución es típicamente de unos pocos segundos. Dado que FAFNER2 toma minutos e ISDEP horas, se considera este tiempo de acoplamiento despreciable.

6.5. Conclusiones

Después de la aproximación teórica al problema, llevada a cabo en las secciones anteriores, aquí la eficiencia del algoritmo de *scheduling* propuesto ha sido demostrada. Siendo DyTSS un algoritmo centrado en un tipo muy

particular de aplicación y hardware -códigos Monte Carlo en infraestructuras Grid-, puede aumentar la eficiencia de la ejecución comparado a algoritmos de uso más general, ya que estos tienen un abanico de recursos más limitado que utilizar.

Además, es importante destacar una ventaja adicional que proporciona Montera sobre los *scheduler* tradicionales, la automatización de tareas: como todas las decisiones acerca del tamaño de los *chunks* a ejecutar y el *scheduling* de tareas son tomadas por Montera, el usuario final no tiene que preocuparse ni emplear tiempo en realizar estas operaciones. Esto representa un paso adelante de cara al usuario final, que ahora puede dedicarse a su área de investigación y delegar el mayor número posible de operaciones no esenciales a esta herramienta software. Además, permite que se utilicen códigos Monte Carlo por usuarios que no son expertos en su ejecución en Grid y desconocen cómo ha de hacerse la planificación de tareas, pues empleando Montera siempre se obtiene el mejor resultado.

Con los códigos de MC siendo ampliamente empleados es muchas áreas diferentes del conocimiento, se espera que una nueva herramienta que simplifica su ejecución en Grid y a la vez aumenta su rendimiento sea bien recibida por la comunidad científica. Además, el hecho de que está siendo desarrollada resolviendo aplicaciones reales y colaborando con usuarios finales garantiza su correcto funcionamiento y usabilidad, gracias a la existencia de un canal de comunicación directo entre los desarrolladores y la comunidad a la que va orientado.

Capítulo 7

Principales aportaciones y trabajo futuro

7.1. Principales aportaciones

Las principales aportaciones de la presente Tesis Doctoral son el algoritmo de planificación DyTSS y el *framework* Montera para la ejecución de tareas en Grid de códigos de Monte Carlo.

El algoritmo DyTSS ha sido diseñado a partir de un algoritmo de *self-scheduling*, teniendo en cuenta las características más distintivas de las infraestructuras Grid: heterogeneidad de los recursos, latencias variables y alta tasa de fallos. Además, el hecho de que haya sido diseñado específicamente para códigos de Monte Carlo ha permitido obtener un gran rendimiento.

En el planificador Montera se han incluido técnicas innovadoras de modelado de la infraestructura y los códigos de Monte Carlo. Se ha demostrado cómo el poseer un mayor conocimiento de los recursos permite un *scheduling* más preciso y efectivo. Además se ha incorporado una planificación en dos niveles que proporciona una gran flexibilidad a la hora de crear nuevas políticas de planificación.

Como se ha demostrado en la sección de resultados, el algoritmo propuesto es más eficiente y robusto que las alternativas existentes hasta el momento. Su combinación con Montera y las herramientas de modelado y *benchmarking* de la infraestructura y códigos de Monte Carlo resulta en una poderosa herramienta, que ya ha sido empleada satisfactoriamente para ejecutar aplicaciones reales en entornos de producción.

Al contrario que la gran mayoría de las alternativas existentes en la bibliografía, DyTSS y Montera han sido ejecutados en infraestructuras Grid reales. Ha quedado demostrado desde un punto de vista teórico y práctico cómo la solución propuesta se adapta a entornos cambiantes y es resistente a fallos.

7.2. Trabajo futuro

El *scheduling* en Grid es un campo donde todavía es posible realizar multitud de innovaciones.

En el presente trabajo se han abordado los códigos de Monte Carlo más sencillos, donde cada uno de los *samples* es independiente de los demás y no es necesario guardar un estado. Sin embargo, hay otras versiones de este método tales como los Random Walks (70) o las cadenas de Markov (71; 72) en las que el estado de un *sample* es necesario para calcular el siguiente.

En este tipo de códigos, la aproximación de Montera no es válida de una manera inmediata, pero parece posible una extrapolación. Por ejemplo, la combinación de un estado inicial aleatorio y un mayor número de tareas con un menor número de iteraciones proporciona resultados esperanzadores, aun a falta de un estudio en profundidad.

Así mismo, el modelado de la infraestructura y del código puede ser aplicado a otro tipo de aplicaciones, tales como el escaneo de parámetros, creando *schedulings* que se beneficiarían de una mayor información disponible. El empleo de una planificación en dos niveles es más difícil de aplicar directamente en este ámbito, aunque sigue siendo una herramienta poderosa con la que contar para determinado tipo de tareas.

Por último, una de las actualizaciones previstas para Montera es la implementación de una interfaz web. A pesar del esfuerzo puesto en crear una aplicación intuitiva, su manejo dista de ser trivial: uso de interfaz por línea de comandos, necesidad de acceder a un equipo remoto vía SSH, empleo de máquinas Linux y demás. Por tanto, la creación de una interfaz web para manejar la aplicación permitiría ampliar la base de usuarios y simplifica las tareas de los actuales. Las propuestas de Parker (137) o Fernández (138) suponen casos de éxito que serán tenidos en cuenta a la hora de llevar a cabo esta tarea.

En cualquier caso, todas estas propuestas de mejora están supeditadas a la utilización actual de la aplicación y las demandas de funcionalidad solicitadas por su incipiente base de usuarios. El empledo de Montera con diferentes códigos de características particulares puede llevar a la necesidad de implementar nuevas funcionalidades. Después de todo multitud de aplicaciones no corresponden completamente a un determinado paradigma computacional, ya que han sido diseñadas para resolver problemas complejos y con requerimientos muy diferentes. Por tanto es de imaginar que Montera deba ser adaptado para poder satisfacer dichos requerimentos y poder abacar un abanico cada vez mayor de códigos con los que es eficiente.

Capítulo 8

Main contributions and future work

8.1. Main contributions

The main contributions of this PhD thesis are DyTSS scheduling algorithm and the *Montera* framework for efficient executions of Monte Carlo codes on the Grid.

DyTSS algorithm has been designed as an evolution of a *self-scheduling* algorithm, bearing in mind the most distinctive characteristics of Grid infrastructures: heterogeneous recourses, variable latencies and a high fault rate. Also, the fact that it has been specifically designed for Monte Carlo codes allows to achieve a high performance.

Montera scheduler has been created with some innovative techniques regarding infrastructure and Monte Carlo codes modeling, as it has been proven that a higher knowledge of the resources leads to a more precise and effective scheduling. It also includes a new two-level scheduling that provides a great flexibility when creating new scheduling policies.

As it has been demonstrated on the results section, the proposed algorithm is more efficient and robust than the existing alternatives. Its combination with Montera and the modeling and benchmarking tools of both the infrastructure and Monte Carlo codes results in a powerful tool. this tool has already been employed to execute real applications on production environments.

Opposite from most of the existing alternatives proposed in the bibliography, DyTSS and Montera have been executed on real Grid infrastructures. It has been demonstrated both from a theoretical and a practical point of view that the proposed solution can adapt to dynamic environments and is fault-resistant.

8.2. Future work

Task scheduling on Grid infrastructures is still an open field, where many innovations can take place.

In the present work only simple Monte Carlo codes have been employed, where every sample is fully independent from the others and there is no data dependance between them. However, there are other versions of this method such as Random Walks (70) or Markov Chains (71; 72) where the result of a sample is necessary to calculate the next one.

On this kind of codes the approach followed by Montera is not straightforward, but an extrapolation may be performed. For example, the combination of a random initial state plus a higher number of simulations with less iterations on each one looks promising, thus a deep mathematical and computational analysis is still missing.

The proposed resource and code modelings can also be applied to other kind of applications such as parameter scan, resulting on schedulers that can profit from a deeper knowledge of the infrastructure. The two level scheduling is harder to directly apply on this area, but it still a powerful tool to be employed on certain kind of tasks

At last, one of the priorities on the developing of Montera is the implementation of a web interface. Although a big effort has been put into creating a simple and easy to use application, its control is still far from being trivial: command line interface, need for accessing a remote server via `ssh`, employment of Linux-based machines and so. Thus, the creation of a web interface to manage the application is expected to increase the number of users and simplify the daily job of the existing ones. Proposals like Parker's (137) or Fernandez's (138) represent success cases that will be regarded as examples when carrying out this task.

Anyway, all these improvements depend on the current usage of the framework and the functionality demands from its incipient user base. The employment of Montera with several codes of different behaviors may lead to the need of implementing new functionalities. After all, many applications do not purely correspond to a simple computational paradigm, due to they have been designed to solve complex problems with different requirements. Thus it is possible to imagine a scenario where Montera has to be adapted to fulfill these requirements and increase the number and nature of codes to which it is efficient.

Apéndice A

El códigos FAFNER2

Se espera que la fusión nuclear sea una fuente de energía sin los problemas medioambientales de las empleadas actualmente, socialmente aceptable y virtualmente inagotable. Para lograr este objetivo, en los dispositivos de fusión se emplean diferentes técnicas de calentamiento del plasma. Éste puede ser calentado usando iones con ondas de radiofrecuencia en el ICRF (*Ion Cyclotron Range of Frequencies*, Rango de Frecuencias del Ciclotrón de Iones), electrones empleando el Calentamiento por ECRH (*Electron Cyclotron Resonance Heating*, Resonancia del Ciclotrón de Electrones) o neutros por calentamiento por NBI (*Neutral Beam Injection*, Inyección de Haz de Neutros).

El método NBI es ampliamente empleado para calentar el plasma en dispositivos experimentales de fusión. Sirva como ejemplo que es uno de los métodos de calentamiento elegidos para el ITER (139). El método es conceptualmente simple: los átomos neutros son capaces de traspasar el confinamiento magnético de un tokamak o stellarator y son ionizados en el plasma por las colisiones con iones y electrones.

En este contexto, FAFNER (124) simula el calentamiento por NBI. Puede ser empleado de manera independiente o acoplado a otros códigos, creando así *workflows* complejos que pueden simular un abanico más amplio de fenómenos.

La simulación por ordenador para optimizar un dispositivo de calentamiento por NBI es también de la mayor importancia para obtener un diseño de eficiencia óptima. Por ejemplo, en el caso del ITER se emplea un código de Monte Carlo tridimensional para analizar el escudo en los conductos, de manera que soporten la radiación nuclear y de *Bremsstrahlung* (140). El único problema es que este tipo de simulación requiere una enorme capacidad de cálculo para producir resultados válidos en un periodo aceptable de tiempo. Para solventar este problema se han empleado diversas soluciones a lo largo de la existencia de la computación de alto rendimiento.

En este apéndice se explicará cómo la aplicación FAFNER2, creada en

los años 80, ha sido modificada y adaptada para poder ejecutarse de manera eficiente en infraestructuras basadas en dos de los paradigmas de mayor relevancia actuales: la computación en clúster y la Computación en Grid. De esta manera, un código que a pesar de proporcionar resultados válidos estaba perdiendo la posibilidad de ser ejecutado en equipos actuales ha vuelto a ser de utilidad en estos entornos.

El código usado en este trabajo, FAFNER2, es la versión adaptada de FAFNER al stellarator helicoidal TJ-II (58). Durante el desarrollo de este trabajo se han creado diferentes versiones de la misma, adaptadas a distintas plataformas.

La primera fase consistió en crear una versión MPI de FAFNER2, capaz de ser ejecutada en clústers basados en la arquitectura X86. A continuación, esta versión fue ligeramente modificada para poder ser ejecutada en Grid, en los *sites* que tienen soporte para MPI. Después, se realizó una versión Serie de FAFNER2, de manera que el soporte de MPI en los *sites* remotos deja de ser necesario y a la vez se produce un aumento de la modularidad de la aplicación. En este punto, la creación de una aplicación DRMAA permitió ejecutar el número deseado de instancias de la versión serie de FAFNER2. Y por último se empleó Montera para ejecutar esta versión Serie, obteniendo así un aumento de rendimiento respecto a ejecutarla de la manera convencional -división del programa en fragmentos del mismo tamaño.

La estructura de este apéndice es la siguiente: en la sección A.1 se define con precisión el problema físico a resolver; a continuación, en las secciones A.2, A.3 y A.4 se plantea la implementación de FAFNER2, tanto desde el punto de vista matemático como su arquitectura interna; después, en la sección A.5 se explicará el proceso por el que las distintas versiones de FAFNER2 fueron creadas. Por último, en el apartado A.6 se expondrán los análisis de eficiencia de las distintas versiones de FAFNER2, con el propósito de poder compararlas y en el A.7 razonar sobre la conveniencia del trabajo realizado.

A.1. La física de FAFNER2

En este apartado se explicará con detalle el problema físico que FAFNER2 resuelve. Para ello se presentará una aproximación sencilla a la fusión nuclear, para después centrarse en el calentamiento NBI. En este caso se detallarán tanto los fenómenos físicos como la manera en que son modelados por FAFNER2, incluyendo las simplificaciones necesarias para que el problema sea abarcable en un periodo de tiempo razonable.

A.1.1. Introducción a la fusión nuclear

La fusión nuclear es el proceso mediante el cual dos núcleos atómicos se unen para formar otro mayor. El nuevo núcleo tiene una masa inferior a la suma de las masas de los dos núcleos que se han fusionado para formarlo y esta diferencia de masa es liberada en forma de energía siguiendo la relación $E = mc^2$.

La reacción de fusión más sencilla es la que une el deuterio y el tritio formando helio y liberando un neutrón, según la ecuación

$$^2H + ^3H \rightarrow ^4H + n \tag{A.1}$$

Los núcleos atómicos tienden a repelerse debido a que tienen carga positiva, de modo que cuanto más cerca están, mayor es la fuerza repulsiva. Sin embargo existe otro proceso, las fuerzas nucleares atractivas -o interacción fuerte-, que son muy intensas a pequeñas distancias. Eso hace que la fusión nuclear sólo pueda darse en condiciones de enorme temperatura (del orden de 300 millones de grados centígrados) y presión. En ese momento, las fuerzas nucleares atractivas superan a las de repulsión y los átomos comienzan a fusionarse.

En los reactores de fusión, esta temperatura y presión se obtienen confinando el material a fusionar en un campo magnético o inercialmente mediante láseres, mientras se va aumentando su temperatura y presión. Bajo estas condiciones, el hidrógeno alcanza el estado de plasma.

A.1.2. El calentamiento por NBI

El calentamiento por inyección de átomos neutros, o NBI por sus siglas en inglés (*Neutral Beam Injection*), es una técnica para calentar el plasma mediante la inyección de átomos neutros de alta energía.

Estos átomos, al no tener carga, pueden atravesar sin problemas el campo magnético en el que está confinado el plasma. Ahí colisionan con las partículas existentes (iones, electrones y átomos fríos), depositando así su energía y pasando ellas mismas a formar parte del plasma.

La fotografía A.1 muestra uno de los tres inyectores del TJ-II del CIEMAT. El esquema A.2 detalla sus partes de un modo gráfico. Su funcionamiento es el siguiente:

1. *Ion Source*. Se genera un plasma frío.

2. Los iones de este plasma se aceleran hasta alcanzar una gran energía, de aproximadamente 40 KeV.

3. *Neutralizer*. Dado que queremos inyectar partículas neutras y no iones, en esta etapa a los iones se les deja sin carga.

Figura A.1: Inyector de partículas neutras del TJ-II del CIEMAT.

4. *Ion Beam Deflection Magnet*. En esta etapa los iones que aún tienen carga son deflectados mediante el uso de un campo magnético, que atrae a las partículas cargadas y las elimina.

5. *Calorimeter*. Esta es una etapa utilizada para comprobar el correcto funcionamiento del inyector. Se coloca un calorímetro que mide la energía de las partículas neutras. Cuando se comprueba que todo es correcto, el calorímetro es retirado para que dichas partículas puedan entrar al reactor.

Los problemas de esta técnica son dos. Por un lado, el hecho de que al calentar el plasma se produzca un aumento de su densidad hace que pueda llegar a producirse un colapso. Además, el que sea una técnica muy costosa hace el uso de simuladores primordial. Esa es precisamente la función de FAFNER2, simular los eventos físicos que ocurren cuando las partículas neutras entran en el plasma.

A.1.3. Descripción de FAFNER2

FAFNER2 es un código diseñado para la modelización de inyección de partículas neutras rápidas en plasmas toroidales tridimensionales y calcula la trayectoria de los iones rápidos resultantes hasta que se pierden en el sistema o son totalmente absorbidos por el plasma.

El código original de FAFNER fue publicado por G.G. Lister, del instituto Max-Planck, en 1985 (124). FAFNER2, una revisión de este software, se

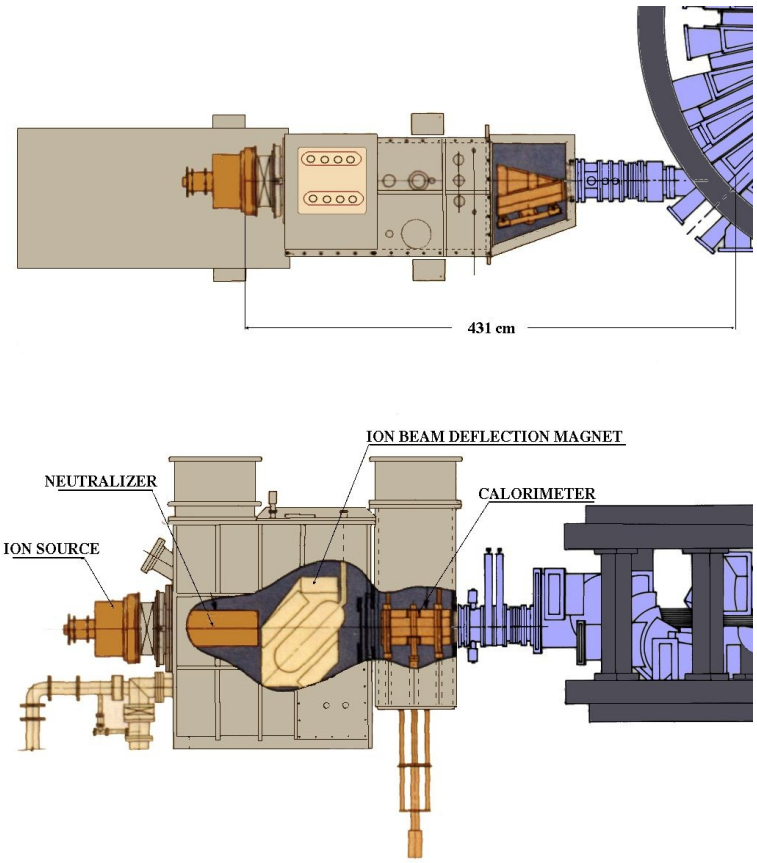

Figura A.2: Esquema de un inyector de partículas neutras.

publicó un año después. Esta versión es una modificación de FAFNER para adaptar su funcionamiento, ecuaciones y variables a la geometría del reactor TJ-II del CIEMAT. Fue creada por el departamento de fusión del CIEMAT en colaboración con los autores originales (58).

La primera parte del código se encarga de simular la inyección de partículas neutras en el plasma. Se obtiene como resultado las coordenadas de dichas partículas y su velocidad. A continuación, en la segunda parte, esta información es usada para determinar la ionización inicial de los átomos neutros en el plasma y, después, la distribución de energía al plasma como resultado de su interacción con los iones rápidos. También se computan en detalle las pérdidas de energía como resultado de las colisiones de las partículas neutras emitidas por cada fuente con los conductos y aperturas.

Desde el punto de vista matemático, se utilizan técnicas de Monte Car-

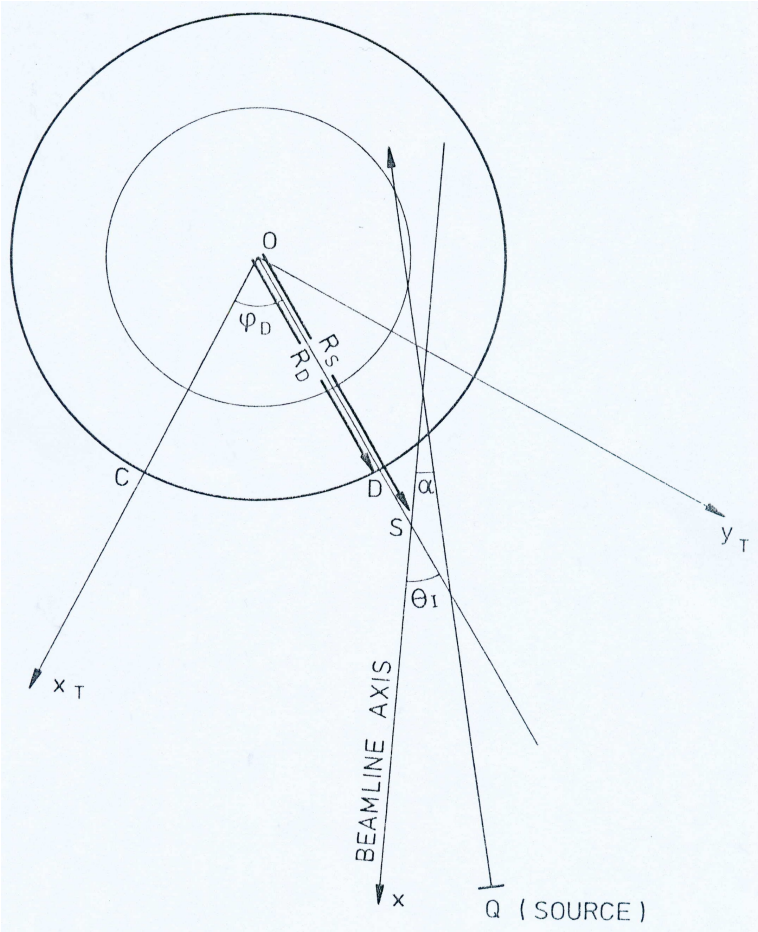

Figura A.3: Sistema de coordenadas del haz: plano medio del toroide.

lo para computar las coordenadas de un conjunto de iones rápidos, correspondientes a las ráfagas de partículas neutras y los parámetros del plasma apropiados para la modelización. Las trayectorias que siguen esos iones y la consiguiente interacción con el plasma son descritos en términos de una ecuación de Fokker-Planck (131). Las trayectorias de los iones rápidos son resueltas en coordenadas de flujo de Boozer (130). Las técnicas numéricas empleadas son bien conocidas y han sido empleadas previamente para describir el calentamiento de los haces neutros tanto en tokamaks como en stellarators.

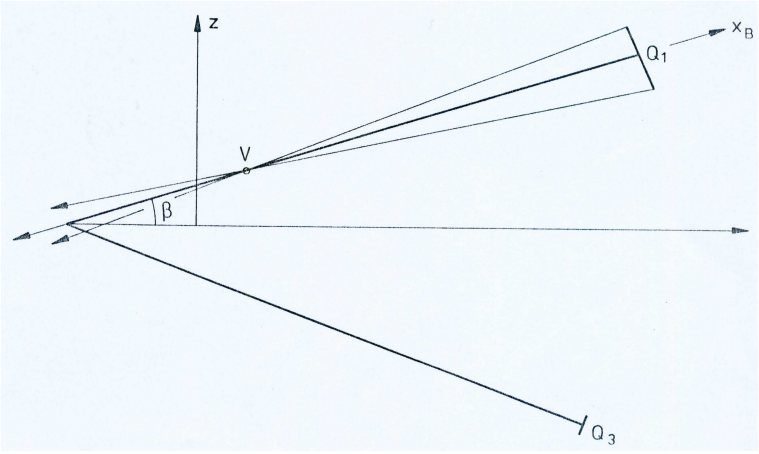

Figura A.4: Vista lateral del haz de partículas.

A.2. Configuración del haz de partículas neutras

A.2.1. Geometría

La configuración del haz de partículas neutras empleada en FAFNER2 está ilustrada en las figuras A.3, A.4 y A.5. Se han adoptado dos sistemas diferentes de coordenadas:

- La geometría del plasma y el toroide son descritas utilizando un sistema de coordenadas cilíndricas *(R, ϕ, z)* con respecto al centro *O* del toroide que contiene el plasma, donde el ángulo toroidal ϕ es medido con respecto a la línea *OC* (normalmente una de las espirales del campo toroidal). El punto de referencia en el toroide para el haz *n* es el centro del conducto de inyección *D*, localizado en *(R_D, ϕ_D)*.
- La geometría de la fuente de partículas neutras de cada haz (Fig. A.3) está descrita usando coordenadas cartesianas *(x,y,z)*, con un origen *S* a una distancia R_s de *O* a lo largo de la proyección de la línea *OD*. El eje *X* está definido en el plano medio del toroide, con un ángulo θ_I a *OS*.

El haz central está dirigido con un ángulo β sobre la horizontal (figura A.4) en el plano XZ, y de tal modo que su proyección en el plano horizontal intersecta el eje *SQ* con un ángulo α (figura A.3) en el plano *X-Y*.

A.2.2. Fuente de iones y *neutraliser*

El código proporciona una opción para detallar la fuente de iones. En el caso que se vaya a inyectar un isótopo de hidrógeno, se emitirán tres especies

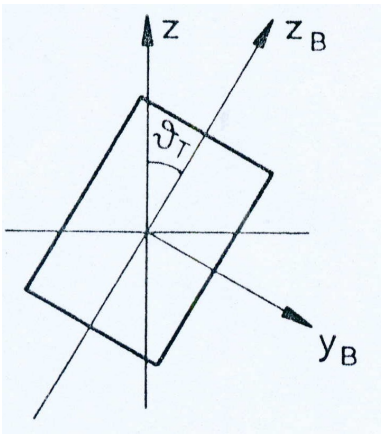

Figura A.5: Inclinación de los inyectores.

de iones, H^+, H_2^+, y H_3^+. Estos iones poseen una fracción de potencia relativa p_1^+, p_2^+, y p_3^+ respectivamente. Así, la potencia máxima disponible de la fuente de partículas neutras, contando con la pérdida de rendimiento durante la fase de neutralización, es

$$P_N(max) = \sum_k P_S p_k^+ n_k \tag{A.2}$$

donde k va de 1 a 3 (para los tres tipos de iones) y asumimos la normalización

$$p_1^+ + p_2^+ + p_3^+ = 0 \tag{A.3}$$

Sin embargo hay cuatro razones por las que se puede producir una pérdida de la potencia disponible:

- Ineficiencia de la fuente de iones.
- Mecanismo de neutralización no ideal.
- Re-ionización entre el neutralizador y el plasma.
- Colisiones de las partículas neutras con los conductos y aperturas.

El último punto es tratado en profundidad en la sección A.2.3, mientras que los anteriores son simulados colectivamente mediante un factor de eficiencia ϵ_k. Por tanto, la potencia suministrada por cada fuente de partículas neutras es:

$$P_N = P_S \sum_k P_S p_k^+ n_k \epsilon_k \tag{A.4}$$

La fracción de energía de cada tipo de partículas en el haz de partículas neutras es

$$p_k^n = \frac{kp_k}{\sum\limits_k kp_k} \tag{A.5}$$

Este parámetro es usado en el código con fines estadísticos.

A.2.3. Conductos y aperturas

Habitualmente es necesario dar al haz una forma determinada antes de que entre en el toroide. Para ello se le hace pasar a través de una superficie sólida en la que se han practicado aperturas con una forma concreta. Un determinado número de partículas colisionarán contra el sólido, quedando el resto con la forma deseada.

La fracción f_L de partículas perdidas en este apartado es tenida en cuenta por FAFNER2, pudiendo elegir entre diferentes tipos y tamaños de aperturas. Con estos datos puede calcularse la potencia total introducida en el toroide, que es:

$$P_I = (1 - f_L)P_N \tag{A.6}$$

A.3. Descripción del plasma

El modelo del plasma consiste en un total de n_p especies diferentes de iones. Asumimos que el plasma está confinado en un toroide de radio r_L, cuyo centro está en R_V.

Las superficies de flujo están definidas por una coordenada radial efectiva s=$\sqrt{\psi/\psi_e}$, donde $2\pi\psi$ es el flujo toroidal y ψ_e es el valor de ψ en el borde del plasma.

Para definir los parámetros del plasma se utiliza un conjunto discreto de n_F puntos, s_J, y cada superficie de flujo j abarca un volumen V_j. El volumen encerrado por una superficie de flujo ψ es definido por una serie de coeficientes $B_V^{(v)}$ tales que:

$$\tilde{V}(\tilde{\psi}) = \sum_{v=1}^{(v)} \tilde{\psi}^v \tag{A.7}$$

El volumen contenido por $s(n_F)$ es el volumen del plasma V_P.

A.3.1. Tratamiento de las impurezas

El número de impurezas puede ser introducido en FAFNER2 en forma de tablas, que deben ser obtenidas de resultados experimentales. En ausencia de

dicha información, se considera que la carga efectiva del plasma es constante.

A.3.2. Densidad de partículas neutras en el plasma

La densidad de átomos neutros en el plasma contribuirá a la pérdida de energía disponible, ya que los iones rápidos pueden ser re-neutralizados en un intercambio de carga y por tanto perderse. La densidad de partículas neutras, n_O, es modelada en el código según la ecuación

$$n_O(s) = n_O(r_L)10^{-\alpha_O(1-s)} \tag{A.8}$$

donde $n_O(r_L)$ es la densidad en el radio máximo y α_O es un coeficiente apropiado.

A.3.3. Magnitud del campo magnético

El campo magnético tridimensional utilizado en los cómputos se define en términos de las coordenadas de Boozer (ψ, ϕ, θ). Su magnitud se define como:

$$B = \sum_{m,n} A_{mn}(\tilde{\psi}) cos(n\phi - m\theta) \tag{A.9}$$

donde $\tilde{\psi} = \psi\psi_e$. Los coeficientes A_{mn} se definen por los polinomios

$$A_{mn}(\tilde{\psi}) = \sum_{v=0}^{3} A_{mn}^{(\tilde{\psi})} \tilde{\psi} v + \frac{1}{2} l(m,n) \tag{A.10}$$

Los valores de *m,n* y *l(m,n)*, junto con ψ_e y los conjuntos de constantes $A_{mn}^{(\tilde{\psi})}$, $B_g^{(v)}$, $B_I^{(v)}$, $B_i^{(v)}$ son leídos de un fichero de datos. Estos coeficientes han sido calculados asumiendo que el Campo Magnético medio en el eje magnético es de un (1) Tesla. El programa incorpora rutinas para ajustarlos según el valor real del campo.

A.3.4. Campos eléctricos

Los campos eléctricos radiales se definen en términos del potencial eléctrico $\Phi(\psi)$ de la siguiente manera:

$$\Phi(\tilde{\psi}) = \Phi(0)(1 - \tilde{\psi}^{m_\Phi})^{n_\phi} \tag{A.11}$$

A.3.5. Descripción del modelo físico

A.3.5.1. Deposición de los iones rápidos

La deposición de iones rápidos en el plasma debido a la interacción con las ráfagas de partículas neutras está gobernada por la ecuación

$$F(s) = \int f(\overrightarrow{x}_i, \overrightarrow{v}_i) e^{-\int_0^x dl/\lambda(\overrightarrow{x}_p, \overrightarrow{v}_i)} d\overrightarrow{x}_i, d\overrightarrow{v}_i \tag{A.12}$$

donde *F(s)* representa el flujo de partículas neutras cruzando la superficie del flujo s, $f(\overrightarrow{x}_i, \overrightarrow{v}_i)$ es la función que modela la fuente de partículas neutras con coordenadas de espacio y velocidad $\overrightarrow{x}_i$ y $\overrightarrow{v}_i$ respectivamente, y λ es el promedio de camino libre hacia las colisiones para todos los procesos de ionización y carga-descarga, que es una función de $\overrightarrow{v}_i$ y de las coordenadas del plasma $\overrightarrow{x}_p$. La integral de dl representa una integración de línea a lo largo de la trayectoria de cada partícula neutra desde su punto de origen a su intersección con la superficie del flujo s, a una distancia x.

El promedio de camino libre hacia las colisiones está definido por:

$$\lambda = v/v_B \tag{A.13}$$

donde

$$v_B = < \sigma v >_e n_e + \sum_k (\sigma_i^k + \sigma_{cx}^k) v n_i^k. \tag{A.14}$$

En esta ecuación, $< \sigma v >_e$ es el coeficiente de las ionizaciones por impactos entre electrones y partículas neutras entrantes, v es su velocidad, n_e es la densidad local de electrones, n_i^k la densidad de especies de iones k y σ_i^k y σ_{cx}^k son las secciones eficaces para los impactos de iones e intercambios de carga entre los iones del plasma y las partículas neutras inyectadas. En esta versión del código, se asume que el plasma está compuesto por distintos isótopos de hidrógeno (con impurezas) y las partículas inyectadas son isótopos de hidrógeno u oxígeno.

Las secciones transversales para el impacto de iones en un plasma de hidrógeno son:

$$\sigma_i = \sum_{m=0}^{6} C_m (ln(E/A))^m \tag{A.15}$$

donde E/A es la Energía por Núcleo y C_m son un conjunto de coeficientes predefinidos.

Las secciones transversales para el impacto con iones de impureza son:

$$\sigma_I = 4{,}6 \cdot 10^{-16} Z_I (1 - e^{-\Sigma)})/\Sigma \tag{A.16}$$

en donde Σ se puede tomar como

$$\Sigma = \frac{E}{3{,}2 \cdot 10^4 A_B Z_I} \tag{A.17}$$

en el caso de que E/A_B sea mayor de 50 KeV por núcleo, siendo A_B el número de iones inyectados y Z_I el número de carga de los iones de impureza del plasma.

En el caso contrario, si E/A_B es menor de 50 KeV se emplea el límite

$$\sigma_I = 4{,}6 \cdot 10^{-16} Z_I \tag{A.18}$$

Las secciones transversales para el intercambio de carga sólo están disponibles en el código si se inyecta hidrógeno, deuterio o tritio en un plasma de isótopos de hidrógeno. En ese caso,

$$\sigma_{cx} = \frac{6{,}937 \cdot 10^{-15}(1 - 0{,}115(log_{10}\tilde{\Sigma})^2}{(1 + 1{,}112 \cdot 10^{-15}\tilde{\Sigma}^{3,3})} \tag{A.19}$$

donde $\tilde{\Sigma} = E/A_B$.

Las partículas neutras que escapan del plasma y colisionan con las paredes del toroide suponen una importante contribución a las impurezas del plasma debido al bombardeo iónico. El código FAFNER2 sólo permite calcular este bombardeo en el caso de que las paredes del toroide sean de hierro o molibdeno, aunque introducir las constantes necesarias para calcular este valor en el caso de otros metales es una tarea trivial. La ecuación que determina el bombardeo iónico es

$$I_S^O = \sum_j \sum_k f_s(j,k) C_s(K, N_I, N_W) I_I(j) \tag{A.20}$$

donde $f_s(j,k)$ es la fracción de potencia de la fuente j, con energía k que viaja por el plasma sin ionizar, $C_s(K, N_I, N_W)$ es el coeficiente de bombardeo e I_I es el coeficiente actual de la fuente j.

A.3.5.2. Deposiciones de energía

La interacción de los iones rápidos con el plasma de fondo, $f(v, \zeta, t)$ en el plasma obedece a

$$\frac{df(v,\zeta,t)}{dt} = \frac{1}{2t_E v^2}\frac{\delta}{\delta v}\{(v^3 + v_c^3)f + \frac{v}{2E_b}(v^3 kT_e + v_c^3 KT_i)\frac{\delta f}{\delta v}\} + v_a \frac{\delta}{\delta \zeta}(1 - \zeta^2)\frac{\delta f}{delta\zeta} \tag{A.21}$$

donde t_E es el tiempo de intercambio de energía de Spitzer y v_c es la velocidad crítica.

Los iones rápidos también pueden cambiar carga con las partículas neutras del plasma y por tanto perderse. Las partículas neutras rápidas resultantes pueden ser re-ionizadas en otra parte del código, pero su efecto no está incluido en esta versión del código.

A.4. Programación de FAFNER2

FAFNER2 está programado empleado Fortran90 (141). Como mecanismo de comunicación entre procesos usa SHMEM (142), que a lo largo de este trabajo fue substituido por MPI 2 (143) en una versión de FAFNER2 y eliminado completamente en otra.

Además, implementa dos librerías con funciones auxiliares, `lib_g3d` y `libfa2_sta`. La primera contiene funciones necesarias para resolver determinadas ecuaciones relacionadas con la geometría 3D del sistema. La segunda, para resolver las ecuaciones explicadas en los puntos anteriores (A.2 a A.3.5), y estudiar la evolución de las partículas a lo largo del tiempo. La librería `libfa2_sta` también emplea SHMEM, por lo que ha sido necesario adaptar su código.

A.4.1. Modularidad de la geometría del reactor

En las primeras versiones de la aplicación FAFNER (124) y su posterior adaptación al dispositivo TJ-II (58) la geometría del sistema estaba embebida en el código, con lo que dicha aplicación sólo era capaz de simular el comportamiento de dicho dispositivo. Posteriormente la aplicación ha sido rediseñada para separar la geometría del reactor en un módulo independiente que se acopla al código principal mediante una sencilla interfaz. De este modo FAFNER2 puede ser empleado para simular la inyección de neutros en cualquier dispositivo que se desee, tanto de tipo stellarator como tokamak.

A.4.2. Resultados

La información que proporciona una ejecución de FAFNER2 es doble.

Por un lado, presenta la posición y velocidad final de un determinado número de partículas. Esto puede representarse con un programa externo de manera gráfica, obteniendo un resultado similar al mostrado en la Figura A.6.

Además, presenta estadísticas varias sobre el estado del plasma. Un ejemplo se puede ver en la figura A.7. La gráfica muestra el porcentaje de partículas en las que ocurrió un intercambio de carga, la potencia absoluta suministrada al toroide, y el porcentaje de partículas que se perdieron.

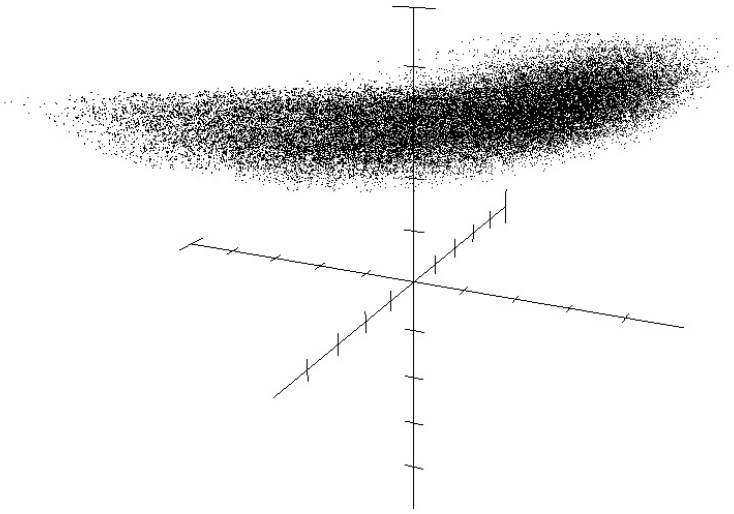

Figura A.6: Representación 3D de los resultados de una ejecución típica de FAFNER2.

A.5. Portado de la aplicación a Grid

En esta sección se describe el proceso por el cual FAFNER2 pasó de ser una aplicación de SHMEM corriendo en procesadores MIPS a ser una aplicación MPI, capaz de ejecutarse en infraestructuras Grid basadas en X86. Además, se describe la creación de una nueva versión Serie de FAFNER2 y su ejecución empleando el *framework* Montera.

El proceso de portado a Grid de FAFNER2 se llevó a cabo en tres pasos.

El primero consistió en la transformación del mecanismo de paso de mensajes, de SHMEM a MPI, dentro de una máquina SGI de memoria compartida. Después el código fue portado a un clúster X86, y por último ejecutado en Grid.

Para la creación de la versión serie usando GridWay se partió de la versión MPI recién descrita. En este caso la aproximación seguida se basó en dos elementos: la eliminación de MPI y serialización del código y la creación de una aplicación Java DRMAA capaz de gestionar la ejecución en Grid de múltiples instancias de la versión serie.

A.5.1. Actualización de la paralelización: de SHMEM a MPI

El primer paso de este trabajo consistió en una transformación en el mecanismo de paso de mensajes, de SHMEM a MPI. Esto es necesario ya que el programa original sólo podía ser ejecutado en máquinas con una ar-

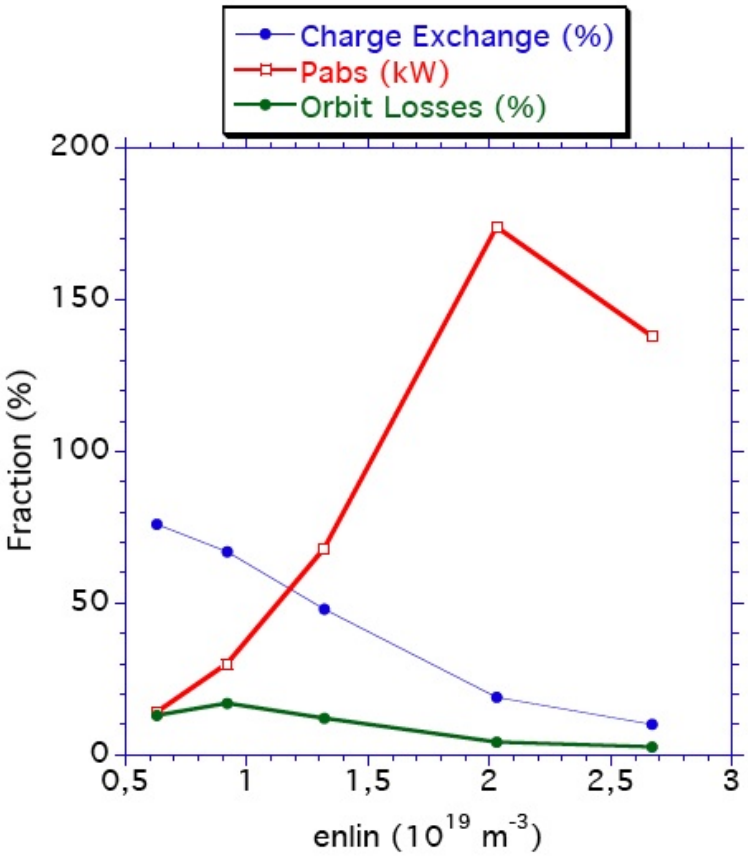

Figura A.7: Estadísticas sobre el estado del plasma proporcionadas por FAFNER2.

quitectura muy específica (equipos de memoria compartida corriendo IRIX o versiones concretas de Linux), lo que limitaba enormemente sus posibilidades de ejecución en Grid.

A.5.1.1. Substitución de funciones y optimización del código

La librería SHMEM está basada en MPI y en muchos casos sus funciones son equivalentes. El primer paso del portado consistió, por tanto, en buscar estas funciones equivalentes y reemplazarlas en el código. En los casos en que esto no pudo ser llevado a acabo directamente, se buscó la función de MPI más cercana y se adaptó el código de manera acorde, para que su funcionamiento siguiera siendo el mismo.

En otras partes de la aplicación, el código fue optimizado evitando llamadas innecesarias a funciones MPI. Para ello se analizó el código y fue posible optimizar el flujo de información, utilizando las funciones MPI más

adecuadas.

A.5.1.2. Entrada/salida

La entrada/salida funciona de manera ligeramente distinta en SHMEM y MPI. A la hora de mostrar información por pantalla y/o escribir en ficheros, fue necesario modificar el código para que el resultado fuera exactamente el mismo. Esto es necesario porque los ficheros generados por FAFNER2 son la entrada de otras aplicaciones y cualquier modificación haría que dejaran de funcionar correctamente.

A.5.2. Actualización de la arquitectura: de SGI a X86

Tras portar el código a MPI, el siguiente paso consistió en portar la aplicación a X86. Y aunque a primera vista esta tarea pueda parecer sencilla (y, de hecho, debería serlo) dista de ser trivial.

Para empezar, existen enormes diferencias entre las distintas versiones de Fortran y MPI. Esto provoca que el código que tiene un correcto funcionamiento con un determinado compilador en un determinado entorno de ejecución, al cambiar de compilador, plataforma o versión de MPI se comporte de una manera completamente diferente.

Por otro lado, un pequeño número de rutinas empleadas en FAFNER2 no pertenecen al estándar de Fortran, sino a librerías implementadas para una determinada arquitectura y sistema operativo. Al migrar la aplicación en ocasiones dejan de estar disponibles, con los problemas que esto provoca. Pero otras veces, aun teniendo el mismo prototipo su funcionalidad es ligeramente distinta, con lo que el funcionamiento de la aplicación se torna errático y da lugar a resultados incorrectos.

Además, cada versión del compilador de Fortran y de las librerías de MPI tienen sus particularidades, dependiendo tanto de la versión como del fabricante. Por tanto es necesario disponer de varias versiones de ambos, y combinarlas hasta que encontrar la forma de que una aplicación concreta -que en otro equipo se podía ejecutar sin problemas- funcione.

Durante la realización de este trabajo, fue necesario el localizar todos estos conflictos e incompatibilidades y corregirlos, de forma que la aplicación tenga el comportamiento esperado. A continuación están detallados algunos de los problemas más significativos que surgieron.

A.5.2.1. Librerías de SGI

FAFNER2 originalmente utilizaba rutinas propietarias de SGI para Fortran. Al migrar el código a X86 dejaron de estar disponibles, por lo que fue necesario implementar estas rutinas de forma manual. Para ello, tras consultar el manual de referencia de estas rutinas (142), se implementaron de

nuevo, empleando el mismo prototipo y teniendo el mismo comportamiento, de manera que su integración fuera inmediata.

La excepción son unas rutinas empleadas en las primeras versiones del programa, y que permitían mostrar la salida de FAFNER2 de forma gráfica. Actualmente los usuarios del programa han dejado de utilizar esta funcionalidad de FAFNER2, empleando programas de terceros más potentes y versátiles. Así pues esta utilidad ha sido dejada de lado, para poder centrar el desarrollo en mejorar la eficiencia de la aplicación.

A.5.2.2. Librería NAG

El CIEMAT posee una licencia de uso de la librería NAG (144), una serie de funciones matemáticas altamente optimizadas que consiguen un alto rendimiento al procesar grandes cantidades de datos. Esta librería sólo está compilada para SGI, por lo que fue necesario recompilarla para X86 a partir del código fuente. Para reducir al máximo su tamaño únicamente se incluyeron aquellas funcionalidades necesarias. De ese modo se pasaron de más de 180 rutinas a únicamente 5, reduciendo su tamaño en un 99 %.

A.5.2.3. Variables de entorno

El tratamiento de las variables de entorno es radicalmente distinto en el equipo SGI de memoria compartida y en el clúster. En la máquina SGI, un programa Fortran MPI podía acceder a estas variables desde cualquiera de sus hilos. Bajo X86 sólo el primer hilo es capaz, por lo que una aplicación que haga uso de estas variables tiene que ser modificada.

Para ello, la aproximación de este trabajo consistió en hacer que, al principio del programa, el primer hilo guardara el valor todas las variables de entorno que se fueran a necesitar en el código en ficheros temporales. Después, en el resto de la aplicación se reemplazaron las rutinas que leen variables de entorno por fragmentos de código que leen los ficheros temporales antes creados.

Dependiendo del tipo de aplicación y el lugar que ocupen en el código las consultas a variables de entorno, también es posible sustituirlas por un *broadcast* desde el primer hilo hacia todos los demás. En este caso se reducen los accesos a fichero a cambio de comprometer la concurrencia. Elegir una opción u otra queda en manos del programador, que deberá evaluar la mejor opción según las características de su programa.

A.5.2.4. De 64 a 32 bits

En el problema físico a resolver (calcular la trayectoria de partículas neutras) la precisión de cálculo es de vital importancia. Por otro lado, dado que el programa realiza múltiples iteraciones sobre cada partícula con el fin

de calcular su posición en diferentes momentos cualquier pequeño error es automáticamente amplificado hasta unos valores inaceptables.

Originalmente, FAFNER2 estaba pensado para ejecutarse en máquinas de 64 bits. Sin embargo, al ser portado a Grid es necesario que pueda ejecutarse en máquinas de 32 bits. Esto ocasiona problemas a varios niveles:

- Es necesario mantener un tamaño de datos, tanto para números reales como enteros, de 64 bits, por lo que hay que obligar al compilador a que ambos tipos de datos sean de 64 bits por defecto. Esto hace que determinadas variables no estén alineadas en memoria. Por ello, hay que modificar el código para evitarlo en la medida de lo posible y, además, hacer que el compilador fuerce el alineamiento.
- El hecho de que se almacenen variables de 64 bits en un computador de 32 bits reduce el rendimiento de la aplicación, dado que el procesador no está optimizado para este tipo de datos. Sin embargo, la posibilidad de utilizar procesadores más recientes y de una potencia mucho mayor compensa con creces este inconveniente.

A.5.3. Portado de la versión MPI a Grid

El siguiente paso en la creación de una versión actualizada de FAFNER2 paralela consistió en habilitar la versión MPI para que pudiera ser ejecutada en Grid. A continuación se describe el proceso seguido, así como una visión de alto nivel de la arquitectura propuesta.

A.5.3.1. Subdivisión del problema

Dado que la finalidad de la Grid es poder lanzar múltiples ejecuciones simultáneas de la aplicación, es necesario subdividir el problema a resolver para que cada ejecución pueda encargarse de una sección.

Ya que FAFNER2 es un código de Monte Carlo, no fue necesario modificar el algoritmo para subdividir el problema. Por el contrario, el trabajo de portado a Grid se centró en subdividir los datos de entrada, para que cada ejecución resolviera una parte. Así, en el presente trabajo se implementaron dos *wrapper*, uno que preprocesa los datos de entrada de la aplicación y otro que hace un post-proceso de los datos de salida.

En primer lugar, en el *wrapper* de preproceso se creó un algoritmo que genera semillas de números aleatorios del mismo modo que hace el programa. De ese modo cada ejecución de la aplicación era inicializada con una semilla distinta, lo que aseguraría que no se están repitiendo los mismos cálculos una y otra vez. Al utilizar el mismo algoritmo de generación que la aplicación original, se consigue que el resultado obtenido sea el mismo.

Tras ejecutar el programa en Grid y recoger los resultados parciales, el *wrapper* de post-proceso se encarga de computarlos para obtener el resultado

final de la ejecución (ver A.5.3.2). Por último genera los ficheros de salida, de manera que son idénticos a los que se obtendrían si el programa se hubiera ejecutado en un clúster local.

A.5.3.2. Resultados parciales

La aplicación FAFNER2 produce datos de salida en dos lugares distintos: a lo largo de la ejecución del programa mediante el hilo número cero y tras juntar todos los hilos. Además, esta información ya está procesada cuando es mostrada en pantalla o escrita en un fichero de salida (por ejemplo, en el caso de una media, ésta ya está calculada). Eso no siempre servía en el portado a Grid, por lo que se modificó la aplicación para que ofreciera estos resultados antes de procesarlos.

Para ello, se introdujeron en el código sentencias que vuelcan en un fichero de datos las variables y cálculos parciales deseados, en forma de XML. Así, además de tener la aplicación funcionando correctamente, se producen resultados que son utilizados en su ejecución en Grid (ver gráfico A.8). Estos ficheros, junto con la salida del programa, constituyen la entrada del *wrapper* de post-proceso. Este *wrapper* lee los ficheros XML y, junto con la información que necesita de los ficheros de salida (que también están en formato de texto plano), genera un nuevo fichero de salida. Este último fichero es equivalente al que se obtendría habiendo ejecutado el conjunto de datos de entrada en un sólo *site*, sin haberlo subdividido y enviado a la Grid.

A.5.4. Versión Serie DRMAA de FAFNER2

Dado que FAFNER2 es un código de Monte Carlo, MPI es empleado únicamente en dividir la simulación entre las diferentes tareas al principio de la ejecución y recoger los resultados finales de todas las tareas una vez que la simulación ha sido completada. En esta sección se describe el proceso para crear una versión Serie que emplea DRMAA (39) preparada para ser ejecutada en Grid.

La figura A.9 proporciona una visión de alto nivel de los pasos descritos a continuación.

A.5.4.1. Aplicación Serie DRMAA

DRMAA (*Distributed Resource Management Application API*, API para Aplicaciones de Manejo de Recursos Distribuidos) (39) es un API de alto nivel que especifica el envío y control de trabajos a uno o más recursos remotos dentro de una infraestructura Grid. La decisión de emplear DRMAA está motivada por las herramientas que provee para controlar el ciclo de ejecución completo de un número arbitrario de tareas: creación, envío al recurso remoto, control de la ejecución y los posibles errores y recuperación de

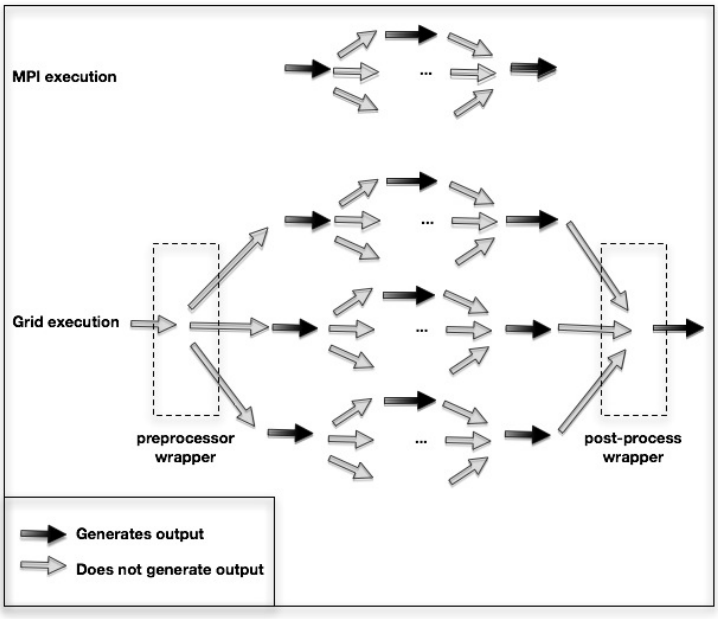

Figura A.8: Generación de los ficheros de salida en las versiones MPI y Grid de FAFNER2.

los resultados. DRMAA permite al usuario implementar aplicaciones distribuidas con una interfaz amigable, ocultando los detalles de bajo nivel de la infraestructura Grid. Además, dependiendo del planificador o metaplanificador empleado, la misma aplicación puede ser ejecutada en recursos privados (SGE, Condor) o en el Grid (GridWay). Una descripción en profundidad de este API puede ser consultada en la sección 2.4.4.

El reemplazo de MPI por DRMAA en FAFNER2 fue llevado a cabo en dos pasos. Primero se eliminó MPI del código, obteniendo así una aplicación en serie. Después se implementó una aplicación Java DRMAA, empleando GridWay para enviar de manera secuencial la versión serie de FAFNER2 al Grid y recoger los resultados.

En la versión MPI de FAFNER2, cuando la aplicación empieza la primera tarea comprueba los datos de entrada, inicializa ciertas estructuras de datos y lleva a cabo cálculos comunes a todas las tareas. Después, la simulación a ejecutar es dividida entre todas las tareas. Al ser un código de MC no hay necesidad de que se comuniquen entre ellas, por lo que cada una es totalmente independiente. Después de que todas las tareas han acabado, la primera recoge todos los resultados parciales y los combina para obtener la salida de la aplicación.

Para emular este comportamiento, el primer paso consistió en desarrollar

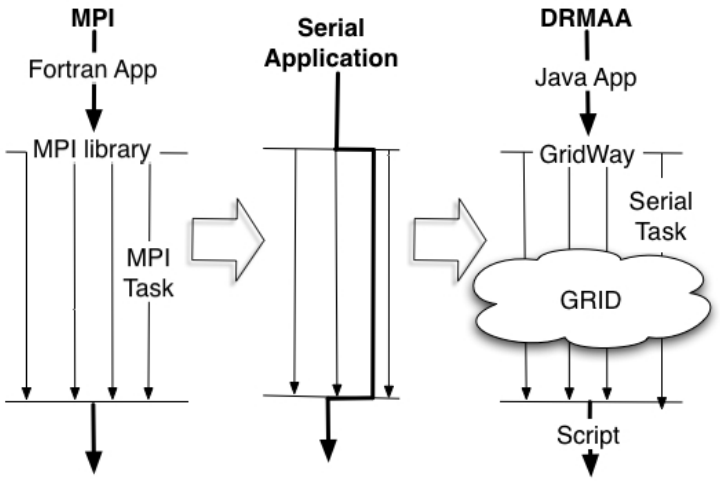

Figura A.9: Arquitectura de las versiones MPI, Serie y DRMAA de FAFNER2.

una versión secuencial de FAFNER2 que se comporta como la primera tarea MPI al comienzo de la ejecución -leyendo los datos de entrada, inicializando estructuras de datos, etc.- y después como una tarea específica de la versión MPI, de manera que pueda llevar a cabo la parte deseada de la simulación. Esta versión secuencial comparte el flujo de ejecución de la versión MPI hasta la primera barrera, el punto donde se lleva a cabo la partición entre las diferentes tareas. Después, se comporta como una tarea MPI cuyo ID se ha recibido como parámetro de entrada, información que antes recibía mediante una llamada MPI.

Cuando las diferentes simulaciones han terminado, la versión MPI de FAFNER2 emplea otra barrera para sincronizarlas todas, recoge los resultados parciales y combina los datos para obtener el resultado final. En esta versión en serie las operaciones de sincronización ya no son necesarias, así que han sido eliminadas del código. En lugar de eso, los resultados parciales de cada tarea son procesados en los distintos *sites* Grid en la medida de lo posible y enviados al recurso local para completar su análisis y procesarlos todos para obtener el resultado final.

Para poder juntar los resultados de varias ejecuciones de FAFNER2, el código tuvo que ser modificado. FAFNER2 produce datos de salida en dos lugares diferentes: a lo largo de la ejecución del programa, y después de llevar a cabo todos los cálculos. Esta información ha sido procesada cuando es mostrada en la pantalla o almacenada en un fichero (por ejemplo, se han llevado a cabo análisis estadísticos) por lo que la aplicación fue modificada para dar los datos sin procesar. Para ello, las variables de entorno requeridas

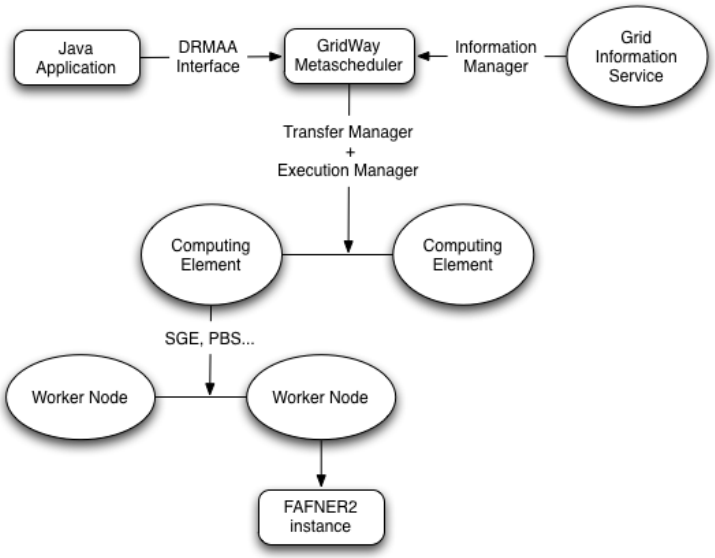

Figura A.10: Descripción de alto nivel de la arquitectura propuesta.

y los resultados parciales son exportados a un fichero XML. Este fichero, junto a la salida del programa, constituye la entrada del *wrapper* de post-proceso. Este *wrapper* lee todos los ficheros XML, y junto con la información de los ficheros de salida, genera el resultado de la simulación completa.

El siguiente paso consistió en la creación de una aplicación DRMAA que emplea la Grid para ejecutar múltiples instancias de la versión secuencial de FAFNER2 de manera simultánea. Esta aplicación DRMAA recibe dos parámetros como datos de entrada: los datos de entrada de FAFNER2 y el número de instancias a ser ejecutado. Con esta información envía los trabajos a la Grid, espera a que todos hayan acabado, recoge los resultados, y los combina para obtener el resultado final empleando el *wrapper* antes mencionado.

Al emplear DRMAA es posible asegurar que el código puede ser acoplado a distintos planificadores y metaplanificadores. En este caso se eligió GridWay. La justificación de su uso puede consultarse en el capítulo 2 (dedicado a la computación Grid), donde sus características y ventajas están explicadas de manera detallada.

La figura A.10 muestra una descripción de alto nivel de la arquitectura propuesta, incluyendo los servicios y herramientas Grid empleados. Sigue la propuesta descrita en la sección 2.1.3, y será brevemente descrita a continuación con propósito de facilitar la labor del lector.

Primero, la aplicación Java anteriormente descrita determina el número y tamaño de las tareas que enviar a la Grid de manera que se cumplan los re-

querimientos del usuario. Después, emplea la interfaz DRMAA para conectar con GridWay, que ejecutará esas tareas. GridWay obtiene la información sobre los recursos remotos a través de GIS (*Grid Information Services*, Servicios de Información Grid), empleando su IM (*Information Manager*, Gestor de Información). Cuando su planificador interno ha decidido dónde ejecutar una tarea concreta, emplea el *Transfer Manager* (Gestor de las Transferencias) para copiar la información al *Computing Element* (Elemento de Cómputo), y el *Execution Manager* (Gestor de la Ejecución) para llevar a cabo la ejecución propiamente dicha. Este *Computing Element* puede ser un nodo aislado, con lo que la ejecución es llevada a cabo directamente, o un gran sistema multiprocesador, como un clúster o una plataforma de memoria compartida. En este caso se emplea un sistema de colas como SGE (*Sun Grid Engine*) o PBS (*Portable Batch System*) para determinar el *Worker Node* (Nodo de Trabajo) donde se ejecutará una instancia de FAFNER2. Por último, cuando la ejecución ha terminado, los resultados viajan en sentido contrario hasta llegar al recurso local.

A.6. Resultados obtenidos

El análisis del rendimiento de las distintas versiones de FAFNER2 que se han obtenido en el desarrollo de este trabajo será llevado a cabo en tres pasos.

- Primero se estudiará la diferencia de rendimiento tras el cambio en el mecanismo de paso de mensajes, de SHMEM a MPI. Para ello ambas versiones de FAFNER2 serán ejecutadas en la misma máquina, bajo diferentes condiciones de carga, con el fin de estudiar su comportamiento.
- A continuación, tras migrar la aplicación de la arquitectura MIPS a X86, se analizará el rendimiento en esta última, para compararlo con los resultados obtenidos en la sección anterior. Este paso es muy importante, ya que FAFNER2 se ejecutará en un clúster local cuando los fenómenos a simular sean de pequeña escala. Con esto quedó demostrado cómo el paso a un clúster de mayor número de nodos, y con procesadores más modernos, disminuye enormemente el tiempo de ejecución.
- Por último, tras portar la aplicación a Grid, se evaluará el tiempo que tarda en ejecutarse en este tipo de infraestructuras. Esto permite comprobar, además de la diferencia de tiempos de ejecución, la escalabilidad del programa. La ejecución en Grid no sólo pretende permitir que el mismo problema sea resuelto en menos tiempo, sino el que sea posible simular problemas más complicados que en un sólo clúster resultaban inabarcables.

Paralelamente se llevará a cabo un análisis de los resultados obtenidos desde un punto de vista físico. El objetivo es demostrar que las diferentes formas de dividir los datos en una ejecución en Grid no afectan al resultado, sino que éste depende únicamente del número total de partículas simuladas.

Por la naturaleza de la aplicación -código de MC puro- esta última fase de comprobación de los resultados obtenidos no debería ser necesaria. Sin embargo, en este trabajo se consideró que, por ser FAFNER2 la principal aplicación empleada para probar los avances obtenidos durante el desarrollo de la presente Tesis, era necesario asegurarse con pruebas tangibles de que todas las operaciones relacionadas con la división de la ejecución en diferentes fragmentos proporcionan resultados igualmente válidos. Además, después del proceso de portado del código y la creación de versiones del mismo muy diferentes, existe la necesidad de cerciorarse de que no ha ocurrido ningún error o imprecisión y los resultados calculados con las distintas versiones son idénticos.

En los siguientes apartados se detallarán los resultados obtenidos en este análisis.

A.6.1. Análisis del rendimiento

FAFNER2 es la principal aplicación empleada durante el desarrollo de la presente Tesis Doctoral para estudiar el funcionamiento de las distintas propuestas con una aplicación real. Así pues, ha sido analizada desde diferentes ángulos, en múltiples infraestructuras bajo distintos paradigmas, de manera que se ha podido determinar estudiar tanto la eficiencia como la validez de los resultados de las diferentes fases del desarrollo.

A.6.1.1. Banco de pruebas

El análisis del rendimiento de las diferentes versiones de FAFNER2 se ha llevado a cabo en un banco de pruebas formado por distintos clústers y la infraestructura Grid EELA-2 (ver `http://www.eu-eela.eu`).

La primera parte de este trabajo, el paso de SHMEM a MPI, se realizó en el computador *Jen50*, un SGI Origin 3800.

Este es un equipo de memoria compartida, con 122 procesadores MIPS R14000 funcionando a 600 MHz. Tiene 126 GB de memoria compartida, y una gran cantidad de almacenamiento tanto en discos como cintas.

Para evaluar el aumento de rendimiento en el cambio de arquitectura (de MIPS a X86) utilizamos dos máquinas. La primera es el *Jen50* antes mencionado, siendo la segunda un clúster heterogéneo llamado *Lince* que está descrito en la tabla A.1.

Dentro de *Lince*, FAFNER2 se ejecutó en los nodos *Serial*. El motivo es que los nodos con Infiniband están reservados para trabajos HPC con mucha comunicación entre hilos, para aprovechar al máximo la capacidad

Tipo de nodos	Características
Infiniband, 32 nodos (64 *slots*)	Dual Xeon 3.2 GHz, 2 GB Hyperthreading Desactivado Soporte Infiniband
Serial, 46 nodos (184 *slots*)	Dual Xeon 3.2 GHz,2 GB 1Gb/s Ethernet
Serial2, 10 nodos (40 *slots*)	Dual Xeon dual-core 3.0 GHz 2 GB 2Gb/s Ethernet (Bonding)

Tabla A.1: Resumen de las características del cluster *Lince*.

Elemento	Características
CPU	144 x Dual Xeon 5450 quadcore 3.0 GHz 96 x Dual Xeon 5450 quadcore 2.96 GHz
Memoria	2GB RAM/core
Red	Doble Infiniband 4X DDR
Almacenamiento	Lustre File Systemm multi-tier RAID 6+0 disk cabinet (60 TB)
Rendimiento	13.8 Tflops (linpack rpeak)

Tabla A.2: Características del cluster *Euler*.

de la red de comunicaciones. FAFNER2, como se ha comentado, tiene poca comunicación entre los hilos, por lo que no es necesario disponer de esta red.

Euler (ver tabla A.2) es un clúster de baja latencia. Asigna diferentes colas a los trabajos a ejecutar dependiendo de sus requisitos y usuario. En la empleada para este trabajo, el máximo número de *slots* por usuario es de 104, lo que establece un límite superior al grado de paralelización. Es el clúster empleado para comparar las versiones MPI en clúster y DRMAA en Grid, por ser el recurso con más potencia de cálculo dentro del CIEMAT.

En el momento de realización de las pruebas (septiembre 2010) La infraestructura de EELA-2 estaba formada por 30 centros de investigación, con más de 3.000 *Computing Elements* y 700 TB de almacenamiento. Esta infraestructura proporciona los recursos suficientes para poder ejecutar la versión FAFNER2 con un alto grado de paralelismo, permitiendo analizar su escalabilidad.

La finalidad de emplear un abanico tan grande de recursos para llevar a cabo el análisis del rendimiento de FAFNER2 es doble. Por un lado, el ejecutarla en distintas infraestructuras correspondientes a distintos paradigmas de computación demuestra la portabilidad de la aplicación y cómo la solución propuesta es capaz de adaptarse a diferentes entornos. Por otro, el

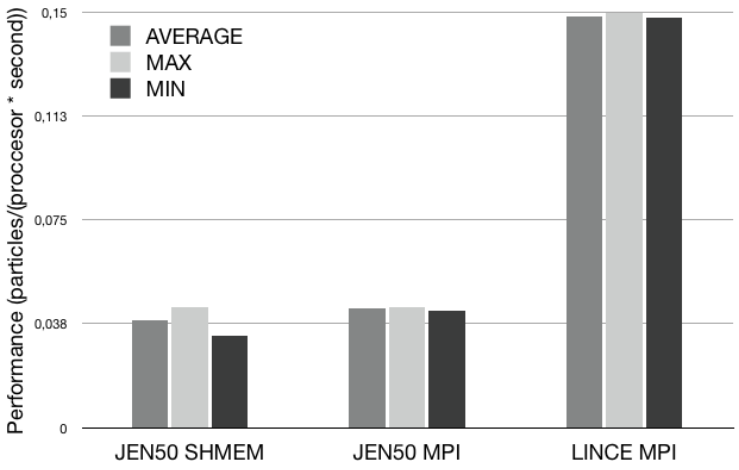

Figura A.11: Rendimiento de las versiones de FAFNER2 de SHMEM sobre MIPS, MPI sobre MIPS y MPI sobre X86.

disponer de un gran número de equipos permite que las comparaciones entre las diferentes versiones de FAFNER2 sean llevadas a cabo de la manera más equitativa posible, eligiendo equipos lo más parecidos entre sí, con lo que se limita al máximo la influencia del hardware en el rendimiento.

A.6.2. SHMEM a MPI

A.6.2.1. Rendimiento de la versión MPI

La medición del rendimiento de la versión MPI se llevó a cabo en dos fases. En la primera se comprobó la mejora al sustituir MPI por SHMEM en un mismo equipo de memoria compartida. Después, se ejecutó la versión MPI en dos equipos diferentes (*Jen50*, de memoria compartida, y *Lince*, un clúster). De este modo puede saberse qué grado de mejora es debido al cambio de paradigma y cuál es debido al cambio de arquitectura.

La figura A.11 muestra de manera gráfica los resultados obtenidos en cada una de estas etapas. Como se puede ver, después de migrar la aplicación de SHMEM a MPI y minimizar su tráfico de red, el rendimiento medio ha mejorado en un 9.89 %. Además, la diferencia entre la ejecución más rápida y la más lenta se ha reducido en un 90.82 %, pasando de 458 a 42 segundos.

A.6.3. SGI a X86

El siguiente paso consistió en actualizar el código de manera que pudiera ejecutarse sobre procesadores X86, no únicamente en los obsoletos MIPS. Tras modificar las librerías y opciones de compilación necesarias, se efectuó

otra medida del rendimiento. Los resultados obtenidos, también incluidos en la figura A.11, demuestran un enorme aumento del rendimiento: el número de partículas simuladas por CPU en un segundo ha pasado de 4.31 a 14.84, un aumento del 244 %. Este aumento demuestra lo necesaria que era la actualización de su arquitectura, habilitándolo para ser ejecutado en equipos actuales.

A.6.3.1. Rendimiento de la versión DRMAA

Con el cambio de paradigma computacional, migrando la aplicación de MPI en clúster a Serie DRMAA en Grid, es necesario llevar a cabo un análisis completo de la nueva aplicación. A continuación se describen los resultados de este análisis desde el punto de vista de la escalabilidad, rendimiento y sobrecarga.

Para ello, se ejecutaron las versiones MPI y DRMAA de FAFNER2 en distintos entornos computacionales: el clúster *Euler* para analizar la versión MPI y la infraestructura Grid EELA-2 para evaluar el rendimiento de la versión DRMAA.

A.6.3.2. Disponibilidad de recursos

Al evaluar las diferencias entre las versiones MPI y DRMAA de FAFNER2, es de vital importancia comprobar el número de recursos donde podrán ser ejecutados.

Como se ha descrito en el apartado anterior, la escalabilidad de FAFNER2 viene únicamente limitada por el número de recursos disponibles. En este caso un recurso es cualquier *site* de la VO *prod.vo.eu-eela.eu*, ya que un trabajo secuencial sin ninguna dependencia de librerías puede ser ejecutado en cualquier *site*. Por otro lado, la versión MPI de FAFNER2 está restringida a *sites* con MPI instalado y su número máximo de tareas viene dado por el número de *slots* del *site* remoto.

La tabla A.3 muestra el estado de la VO *prod.vo.eu-eela.eu* cuando estas pruebas fueron llevadas a cabo (mayo del 2011). La primera columna muestra los *sites* con y sin soporte MPI y las siguientes únicamente los *sites* con soporte MPI. Nótese que en todos los casos se considera cada cola de un *site* como un recurso independiente aunque en ocasiones ocupen el mismo recurso físico, por lo que la capacidad real de la infraestructura puede variar.

Los trabajos enviados por la aplicación DRMAA no necesitan soporte MPI, por lo que pueden ser ejecutados en cualquiera de los 40448 *slots* pertenecientes a los 41 *sites*. Por otro lado, un trabajo MPI necesita *sites* con tantos *slots* como hilos de ejecución se deseen. La tabla A.3 hace patente esta limitación: un trabajo MPI con 10 tareas sólo puede ser ejecutado en 7 *sites*, que se reducen a 1 en el caso de 50 tareas y no hay ningun recurso donde pueda ejecutarse una aplicación con MPI de 100 tareas. Además, co-

Recurso	Serial	10 tareas	50 tareas	100 tareas
Computing Elements	41	7	1	0
Slots Totales	395649	230	84	0
Slots Disponibles	40448	140	0	0

Tabla A.3: Disponibilidad de recursos en la VO *Gisela*.

mo se demostrará en la sección de escalabilidad, es más rápido el encontrar *slots* libres para 50 tareas independientes que un recurso con 50 *slots* libres al mismo tiempo donde ejecutar la versión MPI.

A.6.3.3. Escalabilidad

El primer paso consistió en un análisis de la escalabilidad de la nueva propuesta.

La versión DRMAA de FAFNER2 está diseñada para ser ejecutada en Grid, llevando a cabo experimentos más ambiciosos que los que se pueden ejecutar en clústers. Por lo tanto, es necesario analizar su comportamiento bajo grandes demandas de trabajo. Estas pruebas fueron llevadas a cabo teniendo en cuenta dos parámetros: número de partículas por tarea y número de *slots*.

Para estudiar la escalabilidad en término de número de partículas por tarea se empleó un clúster local, de modo que el entorno es controlado y conocido. El experimento consistió en la ejecución de una sola instancia de FAFNER2 con distinto número de partículas. Los resultados (ver figura A.12) muestran que el tiempo de ejecución y el número de partículas crece en la misma proporción.

Al analizar los resultados es necesario tener dos cosas en cuenta. La primera, que el tiempo de comprobar los datos de entrada y generar los resultados de salida es aproximadamente constante -unos 150 segundos- lo que hace que los tiempos de 10 a 40 partículas parezcan idénticos; la segunda, que la precisión máxima obtenida es de un segundo, por lo que los resultados de la parte inferior de la gráfica no son completamente precisos.

Después se llevó a cabo un análisis de la escalabilidad de FAFNER2 respecto al número de *slots* disponibles en la infraestructura Grid. Los resultados están detallados en la figura A.13 y muestran que el tiempo de ejecución es aproximadamente constante cuando se emplean más de 10 *slots*. Es destacable que los casos anteriores (de 1 a 8 *slots*) obtienen un rendimiento significativamente mayor. Esto es debido al buen funcionamiento del planificador de GridWay: dado que prioriza los mejores recursos para ejecutar un trabajo, los primeros trabajos enviados suelen obtener mejores recursos que el resto.

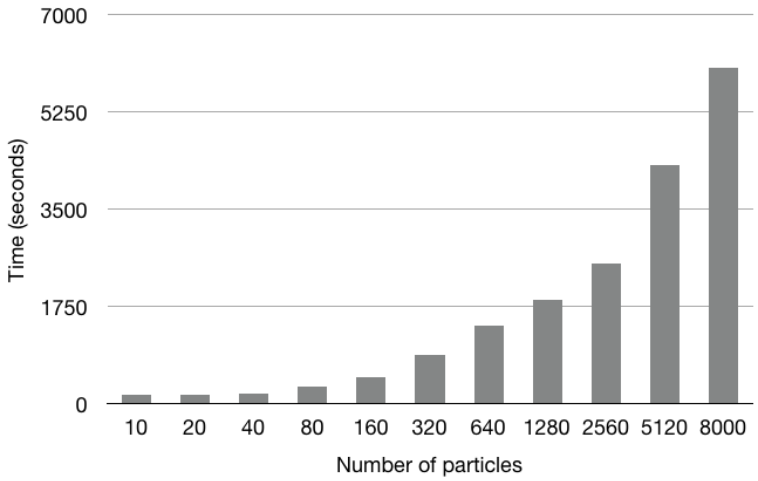

Figura A.12: Escalabilidad de FAFNER2 en número de partículas.

La figura A.14 muestra la escalabilidad de la aplicación en términos de número de partículas con una sola tarea Serie. Como puede verse, la escalabilidad de la aplicación es lineal, lo que demuestra que es escalable tanto en EELA-2 como en *Euler*. Este es el resultado esperado, ya que en los códigos de MC el número de simulaciones repercute directamente en el número de veces que se ejecuta el bucle principal. Esta es una información útil para dividir el problema en subtareas: dependiendo de los recursos y tiempo disponibles, el usuario final puede decidir si prefiere emplear un mayor número de nodos simulando un pequeño número de tareas cada uno, o lo contrario. Habiendo verificado este hecho tanto de manera teórica como experimental, es posible proporcionar al usuario un grado adicional de libertad a la hora de planificar sus tareas. Sin embargo, gracias a Montera y el algoritmo de planificación propuesto en el cuerpo de esta tesis este proceso ya no es necesario, ya que es el propio *framework* el que se encarga de encontrar una distribución óptima de las tareas y llevar a cabo su ejecución.

Si hay una figura que muestre la importancia del proceso de serialización, esa es la A.15: muestra que la escalabilidad de la versión MPI en *Euler* es limitada en términos del número de nodos. Aunque la comunicación entre tareas en FAFNER2 es pequeña, existen dos barreras para sincronizarlas todas situadas al principio y cerca del final de la ejecución, por lo que la aplicación ha de esperar a la tarea más lenta para acabar. El hecho de no tener un uso exclusivo del clúster y las pequeñas diferencias de rendimiento entre las tareas -inherentes a la lógica interna del código- pueden crear un cuello de botella. El resultado analítico muestra que al aumentar el número de nodos de 4 a 512, el tiempo de ejecución únicamente se ha dividido por diez. Hay que señalar que el número de *slots* disponibles fue aumentado a

Figura A.13: Escalabilidad de FAFNER2 DRMAA según el número de tareas enviadas a la Grid.

512 para este test en particular, de manera que se pudiera llevar a cabo el análisis de escalabilidad aquí presentado. Es importante tener en cuenta que aquí fue empleado el clúster *Euler*, de gran rendimiento y altas capacidades de red. Por tanto, es difícil que en un futuro cercano la versión MPI de FAFNER2 pueda escalar más allá del límite aquí encontrado manteniendo este paradigma.

En el caso de la versión Grid DRMAA de FAFNER2 este hecho es irrelevante, ya que los trabajos que se ejecutan son siempre serie. En este caso la escalabilidad está únicamente limitada por la disponibilidad de recursos. Esto permite al usuario simular un número arbitrario de partículas en el número de nodos deseado sin ninguna pérdida de rendimiento.

Además, en el caso de la versión clúster, la figura A.15 muestra la relación entre el número de tareas y el tiempo de cola, lo que limita la escalabilidad del código. Primero, con *Euler* siendo un clúster de 1920 cores, las tareas relativamente breves con un pequeño número de nodos son fáciles de asignar, por lo que su tiempo de cola es corto. Pero después, cuando el número de *slots* requerido crece, el trabajo es más difícil de asignar y su tiempo de cola crece también, a veces hasta alcanzar varios días. Este es un aspecto que el usuario final debe tener en cuenta para llevar a cabo sus experimentos en una cantidad de tiempo razonable y que igualmente puede ser de menor importancia en la Grid a través de los metaplanificadores y la solución aportada en esta tesis.

A.6.3.4. Rendimiento

Para llevar a cabo esta comparativa los recursos elegidos fueron la infraestructura Grid EELA-2 y el clúster de alto rendimiento *Euler*.

La elección del clúster *Euler* en lugar del clúster *Lince* antes empleado

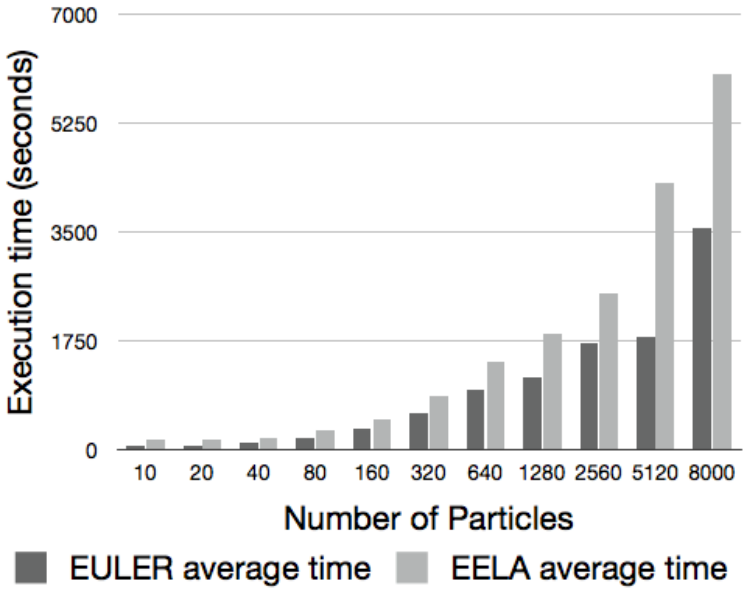

Figura A.14: Tiempo de ejecución en *Euler* y *EELA-2* con un tamaño de problema creciente (1 nodo).

se debe a su mayor rendimiento. El principal problema de *Lince* es que, sin estar obsoleto, corresponde a una generación anterior, mientras que *Euler* había sido adquirido poco antes de la ejecución de esta comparativa, por lo que representa el estado del arte. Se consideró que emplear *Lince*, a pesar de proporcionar resultados más favorables a la solución Grid, supondría que la comparativa no había sido realmente justa y los resultados podrían no ser válidos u obedecer al comportamiento real de la aplicación.

La figura A.16 detalla el rendimiento de *Euler* y EELA-2 bajo una fuerte demanda computacional, compuesta por mil tareas independientes del mismo tamaño. De esta manera es posible comparar el rendimiento y throughput de ambas infraestructuras ejecutando la misma simulación, señalando así las diferencias entre estos dos paradigmas computacionales en un entorno de producción. Para facilitar la legibilidad del gráfico, en el caso de EELA-2, *queue time* agrupa todas las sobrecargas asociadas a la ejecución remota de trabajos: tiempo de planificación de GridWay, tiempo de copia de los ficheros de entrada/salida hacia/desde el *site* remoto y tiempo de cola. En *Euler*, *queue time* representa el tiempo desde que cada tarea fue enviada al planificador local hasta que comenzó su ejecución.

En dicha figura, es destacable que el tiempo de ejecución en *Euler* es aproximadamente constante entre todas las tareas, mientras que en EELA-2 es altamente variable. Esto es debido al empleo de un entorno computacional homogéneo en el caso el clúster y heterogéneo en el caso de la Grid. El hecho

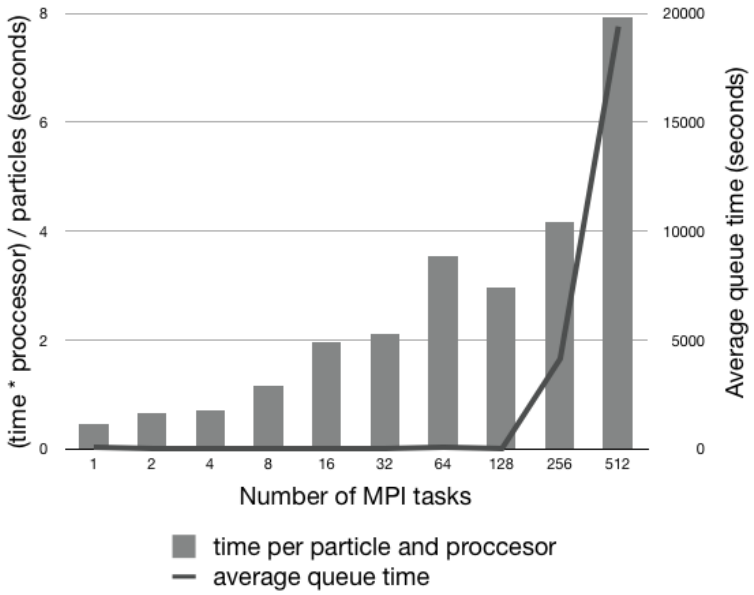

Figura A.15: Tiempo de ejecución en *Euler* con un grado creciente de paralelismo.

de que en *Euler* el tiempo de ejecución sea significativamente menor es debido a la superioridad de sus recursos.

También puede ser observado que el tiempo de cola en *Euler* es una función en escalera, mientras que en EELA-2 es lineal. La causa de esta diferencia es la homogeneidad de los recursos de *Euler* y el límite de 104 trabajos por usuario: las 104 primeras tareas empiezan a la vez y tienen un tiempo de ejecución similar, por lo que finalizan a la vez liberando los recursos, que pasan a ser empleados por las siguientes 104 tareas, repitiendo este ciclo hasta el final de la ejecución. En el caso de EELA-2, al estar compuesta por recursos heterogéneos y dispersos, cada trabajo empieza en un momento diferente.

Además, en el caso de EELA-2 es importante señalar que hay un pequeño número de trabajos con un tiempo de cola enormemente elevado, lo que hace que el *makespan* final crezca. Este problema puede ser solucionado con un ajuste más fino de los requerimientos y condiciones detalladas en la plantilla DRMAA. Un análisis en profundidad de este hecho y las diferentes alternativas para solucionarlo empleando GridWay puede encontrarse en (35). En este trabajo se ha considerado conveniente no realizar estas optimizaciones, para así comparar configuraciones estándar de Grid y clúster. De este modo se puede asegurar que las mejoras obtenidas son fruto del trabajo aquí llevado a cabo y no de un ajuste del software que favorezca una u otra versión.

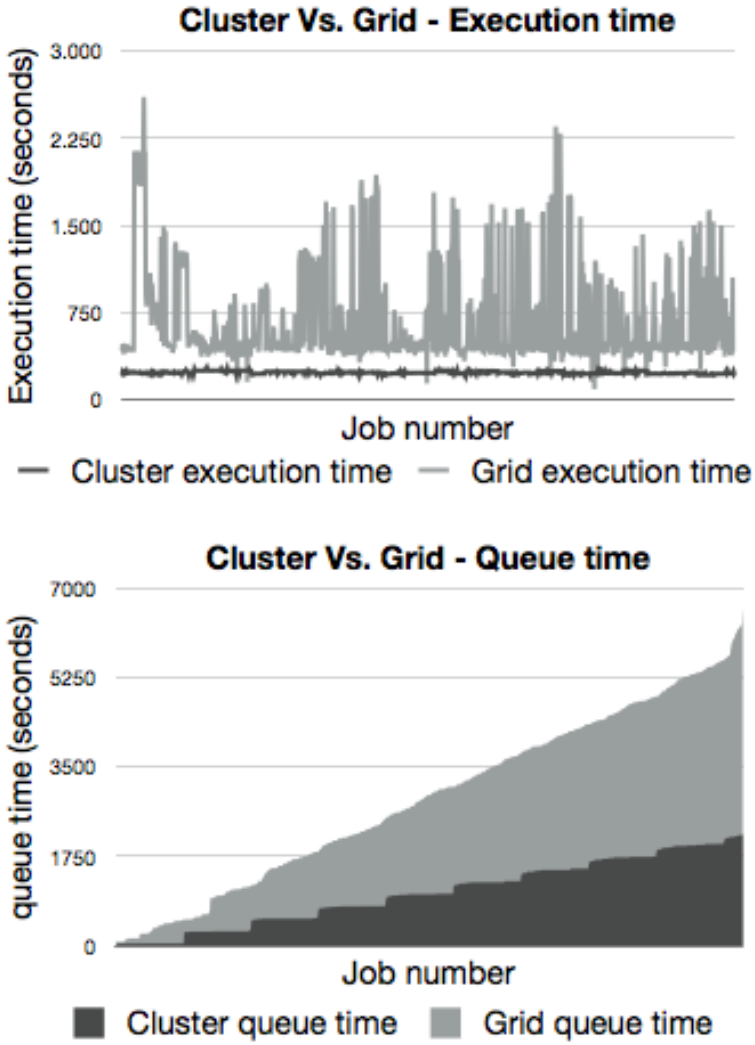

Figura A.16: Tiempo de cola y de ejecución en el clúster *Euler* y en la infraestructura Grid EELA-2.

A.6.3.5. Sobrecarga

Al evaluar el rendimiento de la aplicación, es imprescindible tener en cuenta la sobrecarga inducida por los cambios en el código y por las nuevas capas de software añadidas.

La serialización del código tiene un problema obvio: las operaciones y comprobaciones que antes eran ejecutadas sólo una vez al principio de la ejecución de la aplicación MPI, ahora han de ser ejecutadas en todas las tareas. Esto representa una sobrecarga de aproximadamente 60 segundos por tarea en el clúster *Lince*.

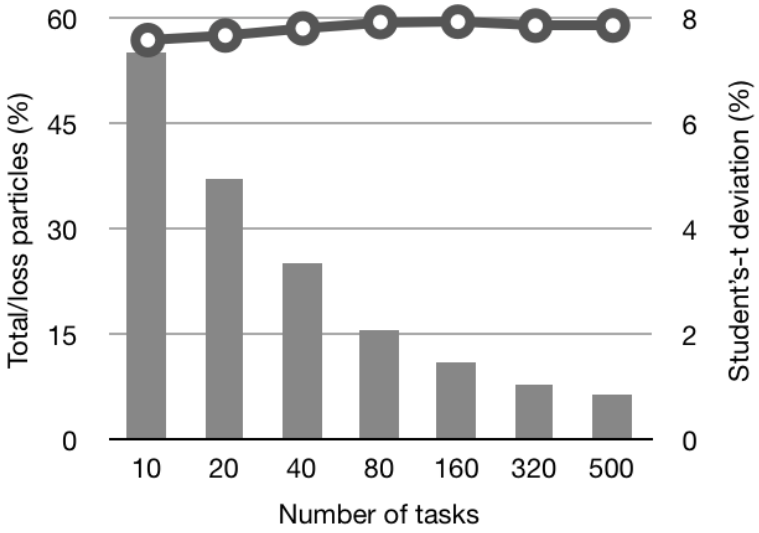

Figura A.17: Evolución del parámetro "Proporción de particulas perdidas/totales" con el número de tareas enviadas (puntos). La precisión de los resultados por medio de la t de Student está también detallada (barras).

La ejecución de FAFNER2 en los *sites* remotos también demanda la copia de los datos de entrada y ejecutable de la aplicación, así como la recuperación de los resultados parciales.

Estas dos sobrecargas escalan linealmente con el número de nodos. Ambas son paralelizables: es posible transmitir varios ficheros simultáneamente y sin pérdida de rendimiento (dependiendo de la configuración de red de los recursos local y remoto) y, una vez transferidos, las ejecuciones en los distintos *sites* son concurrentes. Así pues, su influencia en el resultado final es muy pequeña.

Las sobrecargas producidas en el *site* local también han de ser tenidas en cuenta. La primera es producida por la aplicación DRMAA. En este caso su tiempo de ejecución es de menos de un segundo, con lo cual se puede considerar despreciable. Además el metaplanificador GridWay tiene un tiempo de planificación de 30 segundos -por defecto- y un número de tareas enviadas por ronda de planificación de 15. Esto puede representar un cuello de botella en simulaciones formadas por un número muy grande de tareas pequeñas, pero en este caso no influye de manera significativa en el resultado final.

El *overhead* producido por los *scripts* de post-proceso es más importante. Al ejecutar FAFNER2 con un alto grado de parelización, éste es un factor que ha de ser tenido en cuenta, ya que aumenta linealmente con el número de tareas. Su media es de dos segundos por tarea.

A.6.4. Análisis de los resultados físicos

Llegado este punto ha quedado demostrada la validez desde el punto de vista computacional de las propuestas de este trabajo. Sin embargo, es necesario llevar a cabo un análisis de los resultados obtenidos desde el punto de vista físico, con el fin de comprobar su corrección y precisión.

Para llevar a cabo este análisis, se han comparado los resultados obtenidos ejecutando la aplicación de dos maneras diferentes: primero, se ha simulado un número creciente de tareas con el mismo número de partículas simuladas en cada una; después, manteniendo fijo el número total de partículas, se han dividido en un número creciente de simulaciones variando el tamaño de cada tarea. Estas estadísticas son además útiles para el usuario final, ya que le permiten ajustar con precisión el tamaño del experimento a realizar dependiendo de la precisión deseada: en muchos caso, un cierto número de partículas puede producir un resultado igualmente válido para un experimento en concreto que un cálculo con un número mayor, ahorrando de ese modo tiempo y recursos computacionales.

La figura A.17 muestra los resultados de uno de los parámetros calculados por FAFNER2: la proporción de partículas perdidas/totales. El número de partículas simuladas por tarea permanece constante, por lo que un mayor número de tareas implica un mayor número de partículas simuladas. A medida que se incrementa el número de tareas, la proporción de partículas perdidas/totales alcanza un umbral a partir del cual es estable, dato que puede ser tenido en cuenta por el usuario en sus ejecuciones futuras. Con el objetivo de incrementar el conocimiento estadístico y estimar la precisión de los datos, se ha decidido añadir también la desviación de la t de Student (145) con un nivel de confianza del 95 %.

En la figura A.18 se detalla el resultado de otro análisis. Como puede observarse, la proporción de partículas perdidas/totales calculada por FAFNER2 es usada de nuevo, pero en este caso respecto al número de partículas calculadas por tarea. En este caso se han incluido otros dos valores, el porcentaje de absorción total y la potencia total inyectada.

En este segundo experimento el número de partículas permanece constante en 8000: simular 10 partículas por tarea implica 800 tareas, 20 partículas por tarea implica 400 tareas, etc. De este modo es posible demostrar que la aplicación puede ser correctamente ejecutada en Grid con un grado de paralelismo creciente sin que afecte a los resultados obtenidos. En este caso es destacable que la desviación típica de los resultados se reduce a medida que el tamaño de cada tarea es mayor, y la desviación de la t de Student (de nuevo, con un nivel de confianza del 95 %) crece en una proporción similar. Esto es debido a la naturaleza específica de la aplicación: FAFNER2 devuelve el valor medido de los parámetros mencionados, y después ambas desviaciones son calculadas. Incrementar el número de partículas en cada tarea implica que el valor medio devuelto sea más preciso -por lo que la desviación típi-

ca se ve reducida- pero el tener un número menor de tareas incrementa la desviación de la t de Student. Por tanto, para una evaluación completa del resultado, ambos parámetros deben ser tenidos en cuenta.

Por último, y para completar el análisis de los resultados físicos de FAFNER2, la última parte de este trabajo consistió en llevar a cabo un análisis de la influencia de los números aleatorios. Para ello se reemplazó el generador por defecto de FORTRAN -empleado por la aplicación- por un generador en C. Las diferencias en los resultados obtenidos son despreciables, confirmando así la validez del generador actual.

A.7. Conclusiones

En la realización de este trabajo se lograron tres objetivos fundamentales: un cambio en la herramienta de paso de mensajes, de SHMEM a MPI; portar la aplicación a una arquitectura más moderna, de MIPS a X86; y por último, portar a Grid la aplicación.

La primera tarea, el cambiar la herramienta de paso de mensajes, fue resuelta sin mayores complicaciones, consiguiendo que la aplicación funcione correctamente.

La segunda tarea, portar la aplicación de MIPS a X86, fue la más complicada del trabajo. A los problemas derivados del cambio de sistema operativo y software disponible se unieron los provocados por la utilización de un procesador de 32 bits. Todo esto provocó que fueran necesarios cambios profundos en la aplicación, desde la entrada/salida al generador de números aleatorios (por citar tan sólo algunos). Además, el hecho de que la aplicación no pueda ejecutarse en modo interactivo, sino que haya que enviarla a un sistema de colas *batch*, dificulta enormemente la depuración. Sin embargo, este paso es necesario para poder portar la aplicación a Grid, con lo que no es posible evitarlo.

Por último, se comprobó que FAFNER2 funciona correctamente en Grid. En este caso hubo que desarrollar *wrappers* de pre-proceso y post-proceso, que generan los archivos de entrada necesarios y generan los de salida. Se ejecutó empleando tanto MPI en Grid como DRMAA, analizando el rendimiento de cada una y la corrección de los resultados.

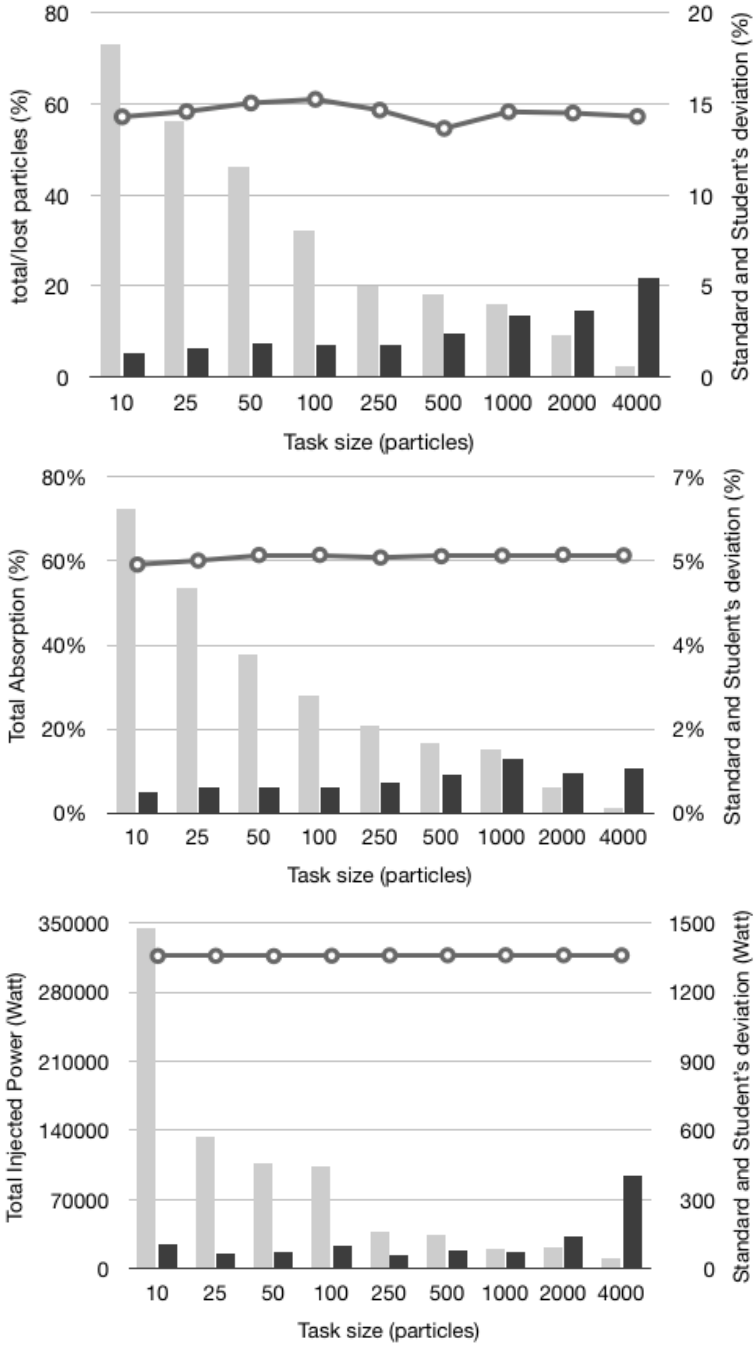

Figura A.18: Evolución de tres parámetros de FAFNER2 con el tamaño de las tareas enviadas (puntos). La precisión de los resultados según la t de Student (gris oscuro) y la desviación estándar (gris claro) también están detalladas.

Apéndice B

El código ISDEP

Como se ha comentado a lo largo de esta Tesis, los códigos de MC son ampliamente empleados para simular diferentes secciones de los dispositivos de fusión. ISDEP (*Integrator of Stochastic Differential Equations in Plasmas*, Integrador de Ecuaciones Diferenciales Estocásticas en Plasmas) (125), es un claro ejemplo de este tipo de aplicación.

Una de las claves en el estudio de los plasmas de fusión es el conocimiento de las trayectorias de los iones en regímenes de baja colisionalidad, de manera que se pueda entender el confinamiento. Esto es válido tanto en el caso de dispositivos de tipo tokamak (146) como en stellarators (147), siendo éste un tema ampliamente estudiado en la literatura.

En este estudio hay varias áreas de interés: el transporte cinético y el comportamiento de las partículas con una configuración magnética determinada; el confinamiento de iones rápidos y partículas alfa; la evaluación de las pérdidas directas; y el efecto de las oscilaciones del magnetismo en el confinamiento de las partículas. Computacionalmente, este problema puede ser abordado por un cálculo de la cinética de los iones basado en técnicas de MC, que resuelven ecuaciones de Langevin en plasmas (148) (149).

Debido a esto, Castejón *et al.* desarrollaron el código ISDEP, inicialmente usado para estimar el transporte colisional de iones para el dispositivo TJ-II sin la necesidad de asumir trayectorias radialmente estrechas para las partículas, como se venía haciendo en el estudio de transporte neoclásico. En este trabajo, se seguían hasta un millón de partículas en una configuración magnética realista (150) en el dispositivo heliac mencionado ($R = 1.5$ m, $a < 0.22$ m), teniendo en cuenta las colisiones y el campo magnético. Como resultado, era posible obtener el incremento monótono del calor y flujo de las partículas con menor radio, el carácter no difusivo del transporte, la aparición de asimetrías en las superficies magnéticas y el carácter no maxwelliano de las funciones de distribución. En las figuras B.1 y B.2 se muestran dos de estos resultados de manera gráfica.

Posteriormente, y también mediante el uso de ISDEP, se pudo confirmar

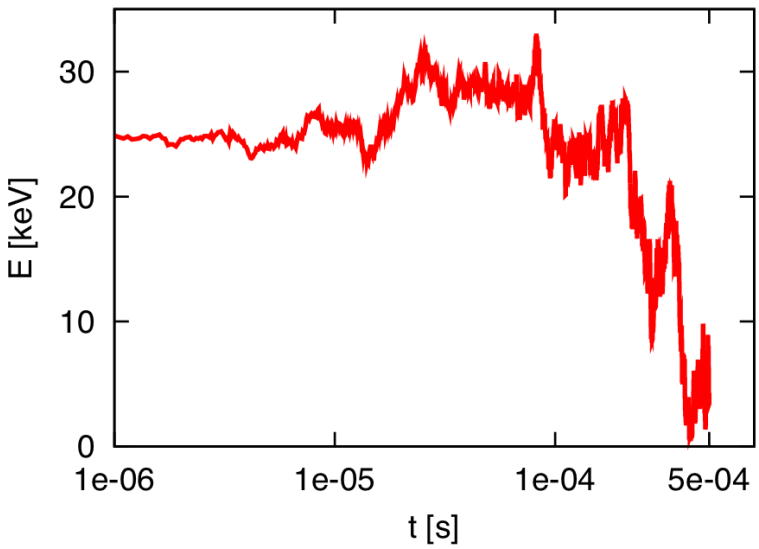

Figura B.1: Evolución de la energía de un ión calculada con ISDEP.

que el incremento de la temperatura de los iones en el calentamiento electrón-ciclotrón (régimen Core Electron-Root Confinement, CERC) es debido a la acción conjunta de la transferencia de energía electrón-ion y a un aumento del confinamiento de los electrones (151).

En todo caso, el desarrollo de ISDEP aun no está cerrado, pudiendo incorporarse distintas mejoras. A medida que evolucione se espera un uso enorme del código y la computación Grid puede ofrecer una solución apropiada.

B.1. La física de ISDEP

Según algunas aproximaciones, el plasma puede ser descrito con una FPE (*Fokker-Planck Equation*, Ecuación de Fokker-Planck) (129). Esta FPE es una derivada parcial en 5 dimensiones para la función de distribución del plasma, imposible de resolver matemáticamente y muy difícil de resolver numéricamente. ISDEP (125) está basado en la equivalencia entre la FPE y las ecuaciones de Langevin (131). Las ecuaciones de Langevin son ecuaciones diferenciales estocásticas para el movimiento de una sola partícula, en lugar de para todo el plasma.

El desarrollo del código ISDEP fue realizado para resolver las ecuaciones del centro guía en presencia de colisiones con iones. La aproximación GC (*Guiding-Centre*, Centro Guía) (152) es empleada para separar el giromovimiento rápido del desplazamiento relativamente lento de la partícula dentro del dispositivo. En lugar de considerar la posición y velocidad de la partícula física, se tienen en cuenta la posición y velocidad del centro guía de su

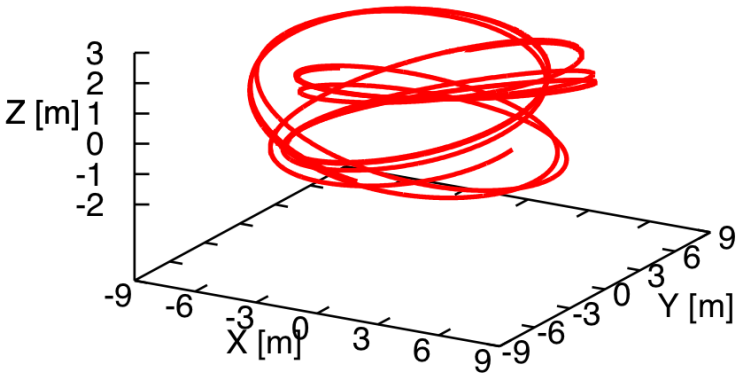

Figura B.2: Trayectoria de un ión calculada con ISDEP.

precesión de Larmor. La dinámica de los iones se representa con un conjunto de cinco SDE (*Stochastic Differential Equation*, Ecuaciones Estocásticas Diferenciales), que son:

$$\frac{\partial \vec{r}_{GC}}{\partial t} = \vec{a}_{r_{GC}} \tag{B.1}$$

$$\frac{\partial \lambda}{\partial t} = [a_\lambda + a_\lambda^{Ito}] + b_\lambda \xi_\lambda \tag{B.2}$$

$$\frac{\partial x^2}{\partial t} = [a_{x^2} + a_{x^2}^{Ito}] + b_{x^2} \xi_{x^2} \tag{B.3}$$

Donde las magnitudes ξ dependen ambas del ángulo de *pitch* (λ) y la energía cinética (x^2) representa ruidos "blancos" independientes. El espacio quinta-dimensional está compuesto de la posición del centro guía de la precesión de Larmor, r_{GC}, el *pitch* λ y la energía cinética normalizada de la partícula x^2:

$$x^2 \equiv \frac{v^2}{v_{th_i}^2} \tag{B.4}$$

$$v_{th_i} = \sqrt{\frac{2kT_i}{m_i}} \tag{B.5}$$

$$\lambda \equiv \frac{v_\parallel}{\sqrt{v^2}} \tag{B.6}$$

Donde m_i es la masa de los iones, T_i la temperatura y k la constante del Boltzmann.

Como se explica en (125), este sistema representa un proceso de difusión de Itô de iones de prueba con un plasma de fondo y puede ser resuelto linearizando la ecuación del tipo Fokker-Planck para la función de distribución

fase-espacio quinta-dimensional $f(t, rGC, \lambda, x^2)$, que describe la evolución de todos los iones en el plasma:

$$\frac{\partial f}{\partial t} + \vec{\nabla}_{r_{GC}} \cdot (\vec{u} f) + \frac{\partial}{\partial t}(a_\lambda f) + \frac{\partial}{\partial x^2}(a_{x^2} f) = \mathcal{L} f \tag{B.7}$$

La linearización se lleva a cabo considerando despreciable la dependencia del operador diferencial L en el plasma de fondo. Así, f representa un conjunto de iones de prueba moviéndose en presencia de un fondo de iones de campo que determina a_i. Al principio del proceso, las partículas de campo son descritas por su densidad experimental, temperatura y potencial electrostático de sus perfiles y electrones.

El trabajo de Velasco (151) añadió un nuevo concepto. Para estimar la transferencia de calor de los electrones a los iones, el operador de colisión fue extendido siguiendo la propuesta de Chen (153), para poder tener en cuenta las colisiones de los iones de test con los electrones. Esta aproximación sigue siendo válida cuando se incluyen otras especies en la distribución de partículas de campo. Obviamente $\mathcal{L}$ ahora ha de incluir los términos de colisión con los electrones, lo que no implica añadir términos de difusión (ver apéndice A en el artículo de Velasco (151) para obtener los detalles).

La función de probabilidad para las posiciones iniciales y velocidades de los iones de test de acuerdo a la distribución del campo iónico subyacente $f(t, rGC, \lambda, x^2)$ se asume que es Maxwelliana en la energía cinética, es decir:

$$f(t_0, \vec{r}_{GC}, \lambda, x^2) = n(\vec{r}_{GC}) \frac{x \exp(-x^2)}{\sqrt{\pi}} \tag{B.8}$$

Donde $n(r_{GC})$ es la densidad numérica de los iones de fondo. Por tanto, la densidad probabilística de su posición y velocidad en un tiempo t es proporcional a $f(t, rGC, \lambda, x^2)$. El principal problema de esta aproximación es que el calentamiento de los iones no puede ser estimado usando un perfil de temperatura de los iones independiente del tiempo debido a que el momento de transferencia de energía ion-electrón ocurre unas cien veces más tarde que el de la colisión ión-ión.

Para efectuar el cálculo de Langevin de la función de distribución se consideran los siguientes pasos: en un campo independiente del tiempo f_0^{field} se calculan un gran número de iones de test hasta que $t = 0{,}1$ s. A partir de estas medidas se obtiene la distribución de iones dependiente del tiempo f_0^{test}. En la siguiente iteración, se calculan nuevas órbitas de iones en presencia de un campo iónico dado por $f_1^{field} \equiv f_0^{test}$. Con estas trayectorias es posible medir f_1^{test}. Después de varias repeticiones de este esquema se obtiene una solución estacionaria (dependiente del tiempo), por lo que la función de distribución de los iones de test evoluciona en equilibrio con la del baño

termal bajo la influencia de campo eléctrico y los iones de campo, es decir, $f_N test = f_N field \equiv f_{N-1}^{test}$ cuando $N \to \infty$.

Es posible encontrar más detalles acerca del proceso computacional y los fenómenos físicos asumidos (fuera del alcance de este trabajo) en (125; 151), incluyendo detalles sobre la integración de largas trayectorias en la compleja configuración magnética del TJ-II, pero el sistema descrito en este trabajo representa básicamente la simulación de MC. El análisis estadístico de muchas trayectorias de partículas es empleado en ISDEP para obtener los parámetros físicos del plasma completo, tales como energía, tiempo de confinamiento, puntos de escape de los iones, etc.

B.2. Portado de la aplicación a Grid

Como ha sido previamente mencionado, el continuo desarrollo del código ISDEP proporciona la posibilidad de enfrentarse a problemas nuevos y más ambiciosos. Además, su desarrollo en diferentes plataformas computacionales ha permitido el ampliar su uso e impacto. En este aspecto, la tecnología Grid supone una potente herramienta: nuevas características como la adaptación del código al ITER, mantener los términos de temperatura no lineales o la introducción de las resonancias del Alfvén en el código pueden ser realizadas empleando computación Grid.

La primera versión de ISDEP preparada para ser ejecutada en Grid fue portada dentro del proyecto EGEE. Después, una nueva versión -adaptada para tokamaks- fue portada dentro del marco de EUFORIA (154). En el segundo caso se decidió incluir un término de temperatura no lineal, esto es, la interacción cuasi-linear onda-partícula que causa el calentamiento del plasma, lo que permitió analizar el calor y transporte colisional al mismo nivel. Finalmente ISDEP ha sido portada a dispositivos de computación voluntaria tipo BOINC (132).

B.2.1. El problema del post-proceso

A pesar de que el proceso de portado a Grid de ISDEP no supuso un reto especialmente complicado, su adaptación a Montera implicó ciertas modificaciones del código que es conveniente destacar.

Después de que todas las ejecuciones de ISDEP son completadas, se emplea una aplicación de post-proceso para analizar los resultados parciales proporcionados por ISDEP y obtener el resultado final de la aplicación. En este sentido ISDEP obedece directamente al esquema propuesto en la figura 3.3, relativa a la implementación de códigos Monte Carlo para aplicaciones distribuidas.

Con el propósito de reducir al máximo las necesidades de computación en el recurso local, la mayoría del procesamiento es ejecutada en los *sites*

remotos. De este modo el análisis de los datos obtenidos es pararelizado en la medida de lo posible, además de limitar al máximo el volumen de información que ha de ser recuperado en archivos de salida por parte del recurso local.

El problema de esta aproximación es que la aplicación de post-proceso original esperaba que cada instancia de ISDEP tuviera el mismo tamaño, lo cual no es necesariamente cierto al utilizar las técnicas de *scheduling* aquí propuestas. Por tanto, esta aplicación tuvo que ser modificada para proporcionar la flexibilidad deseada, y los resultados parciales proporcionados por ISDEP ligeramente incrementados para permitir un análisis más complejo.

Esta situación es análoga a la experimentada en el caso de FAFNER2 y descrita en el apéndice A.5.3.2, y la solución es básicamente la misma: encontrar la información necesaria para llevar a cabo el procesamiento de los resultados parciales; modificar la aplicación remota para que proporcione esta información; y ejecutar el resto del procesamiento en los *sites* remotos.

B.3. Conclusiones

ISDEP es un código de fusión MC con una gran demanda computacional dedicado a solucionar las dinámicas del plasma en un dispositivo de fusión. Puede ser acoplado a diferentes códigos para estudiar una abanico más amplio de fenómenos del plasma. Esto, junto a su naturaleza muy específica y un cuidado diseño que permite su ejecución en plataformas distribuidas de una manera sencilla y eficiente, hace que sea una de las herramientas más prometedoras dentro de la comunidad de fusión.

En este escenario, una distribución eficiente de las tareas supone una valiosa mejora, al permitir a los investigadores el llevar a cabo estudios más ambiciosos y complejos en un periodo de tiempo más corto. Por eso ISDEP ha sido portada a diferentes plataformas distribuidas, donde su ingente necesidad de recursos puede ser satisfecha en un tiempo aceptable.

Apéndice C

El código FastDEP

En los apéndices A y B se han descrito con detalle las aplicaciones de fusión FAFNER2 e ISDEP, orientadas al calentamiento NBI y la trayectoria de iones respectivamente.

ISDEP (125), el Integrador de Ecuaciones Diferenciales Estocásticas en Plasmas, es una aplicación que permite estimar el transporte colisional de iones en dispositivos de fusión.

La posición de dichos iones al comienzo de la ejecución es suministrada a ISDEP en un archivo binario que constituye los datos de entrada. Dicho fichero contiene información relativa a las coordenadas espaciales, velocidad y *pitch* de una serie de partículas. ISDEP selecciona aleatoriamente un subconjunto de estas partículas y simula su movimiento en el interior del plasma durante el tiempo deseado.

Hay varias maneras de conseguir el conjunto de partículas que se empleará como datos de entrada. De entre ellas una especialmente interesante es el empleo de FAFNER2 (124) que, como se ha descrito en el apéndice correspondiente, simula la inyección de partículas neutras en el plasma hasta que son ionizadas o escapan al exterior del mismo.

El propósito original de FAFNER2 es obtener información relativa al conjunto de las partículas simuladas, tal como su temperatura, el porcentaje de ionizaciones o cómo su inyección ha afectado al estado del plasma. Para ello analiza el estado final de las partículas y calcula diversas estadísticas, que constituyen los datos de salida de la aplicación. Sin embargo es posible emplear el estado de estas partículas tras la simulación como la entrada de ISDEP, de manera que se analiza su evolución temporal en una ventana mucho más grande de tiempo, lo que permite un análisis más profundo del dispositivo de fusión.

La paralelización llevada a cabo con FAFNER2 incluye la división del problema en subproblemas independientes y la posterior recolección de los datos de salida de estos subproblemas para su análisis conjunto y obtención de los resultados finales. Para ello, cada tarea enviada al Grid simula un

número determinado de partículas y devuelve los resultados sin procesar, que son analizados por el llamado *wrapper de post-proceso* en el nodo local.

En caso de que se deseen los resultados completos de la ejecución de FAFNER2, este proceso es inevitable. Pero si lo que se desea es únicamente obtener las posiciones y velocidad de cada partícula para emplearlos como entrada en ISDEP, el problema puede ser abordado de manera diferente: el resultado de cada subproblema de FAFNER2 es utilizado como la entrada de un subproblema de ISDEP, no siendo de este modo necesaria la fase de recopilación de datos de FAFNER2. Más aún, ambas tareas pueden ser ejecutadas de manera conjunta en un mismo sitio Grid, con lo que los *overhead* relacionados con este paradigma de computación se ven reducidos y el rendimiento del conjunto aumenta.

En el siguiente apéndice este proceso será descrito con detalle, tanto desde un punto de vista computacional como físico: el acoplamiento de ambos códigos permite abrir la puerta a nuevos análisis de los dispositivos de fusión **??**, siendo actualmente empleado en el TJ-II con una metodología que permite exportarlo al LHD o ITER de manera natural. Además, el hecho de que ambos sean códigos Monte Carlo hace el empleo de Montera enormemente interesante, más aun cuando su capacidad de acelerar cada uno por separado ha sido previamente comprobada.

C.1. La física de FastDEP

Es posible acoplar FAFNER2 a otros dispositivos de fusión al emplearlo como datos de entrada. Este es el caso del *workflow* presentado aquí (FastDEP), que abre la puerta a la exploración de un nuevo abanico de problemas físicos.

Con FastDEP es posible estimar las posiciones iniciales de los iones rápidos en el plasma, y posteriormente seguir las trayectorias de los iones en el plasma. Este cálculo produce un mapa tridimensional de los puntos de colisión de los iones rápidos con la cámara de vacío del dispositivo, que en nuestro caso de uso es un heliac stellarator. De hecho, el *workflow* presentado permite el seguir a partículas neutras que se convierten en iones cuando colisionan con el plasma presente en el dispositivo.

El calentamiento y alimentación del plasma son de utilidad para la corriente motriz y momento al modelar el proceso de NBI, por lo que la función de distribución de iones rápidos, considerada como la perturbación de un plasma estático preexistente, puede ser ahora cómodamente calculada numéricamente, ya que el calentamiento y la eficiencia de la corriente motriz dependen fuertemente de los parámetros de transporte de iones rápidos, que pueden ser también calculados con FastDEP. Las propiedades de transporte de iones colisionales en el dispositivo completo pueden ser estudiadas junto a los puntos de contacto de los iones que se escapan del plasma y la pared

de la cámara de vacío.

Otro aspecto del nuevo código es que los iones rápidos están en un régimen de baja colisionalidad, por lo que el problema de encontrar una solución numérica al transporte de iones rápidos está superado, incluso cuando la complejidad del campo magnético en dispositivos 3D esté también presente. Más aun, usando la función de formalismo de Green (155), la función continua de distribución puede ser también calculada al comienzo de una descarga, cuando el calentamiento y alimentación del plasma no son tan importantes.

El estudio de la rotación toroidal y su influencia en el confinamiento también pueden ser obtenidos. Este es un aspecto clave en los tokamaks, donde la rotación toroidal puede juzgar un rol crucial en la estabilización magnetohidrodinámica (156) y donde la rotación espontánea, sin ninguna entrada externa de momento angular, es observada (157). La corriente causada por los iones rápidos puede contribuir significativamente a la corriente total del plasma, modificando por tanto el confinamiento magnético. Incluso aunque la rotación toroidal sea limitada y los iones rápidos son -en principio- la única fuente de momento en los dispositivos tipo stellarator, estudiar este fenómeno es también de importancia a la hora de validar los modelos propuestos cotejándolos con datos experimentales. Más aun, la rotación toroidal en los stellarators puede también jugar un rol relevante en la estabilización del plasma (158) y puede aparecer simultáneamente, sin ninguna entrada de momento (159). La rotación poloidal es también relevante para el confinamiento, ya que se ha demostrado que los flujos poloidales pueden ser beneficiosos para el confinamiento tanto en tokamaks como en stellarators.

Otra posibilidad que proporciona FastDEP es la determinación del tiempo de frenado de los iones de NBI, ya que el proceso puede ser ajustado con una función de decaimiento exponencial. Calcular esto es una manera de conocer la eficiencia del proceso de calentamiento: cuanto menor es este tiempo, más eficiente es la absorción de potencia por el plasma y menores son las pérdidas de iones rápidos.

La distribución de escape y confinamiento también pueden ser estimados por medio de FastDEP. El confinamiento es descrito por la persistencia de las partículas y por el cono de pérdidas en el espacio de velocidad, siendo la persistencia de las partículas la posibilidad de encontrar una partícula en el plasma después de un tiempo dado (proporciona una medida del tiempo de confinamiento de las partículas) y el cono de pérdidas la región del espacio de velocidad donde los iones son perdidos en un intervalo de tiempo dado. La determinación de estas dos características del confinamiento de iones rápidos es independiente de los cálculos del estado estacionario y muestran el efecto de una configuración magnética particular y las propiedades de inyección en los dispositivos de fusión, de modo que es posible optimizar su construcción.

Por último, el *workflow* presentado puede ser fácilmente aplicado a otros dispositivos de fusión y, de esta manera, las simulaciones pueden ser com-

paradas con las medidas experimentales en el proceso antes mencionado. El empleo de coordenadas cartesianas hace que se puedan incluir geometrías con islas magnéticas o zonas ergódicas, así como el volumen de la capa de hendiduras donde la líneas de campo son abiertas. Además, el diseño de la aplicación hace que la geometría relacionada con cualquier instalación sea cargada en un módulo independiente, quedando así encapsulada y aislada del resto de la aplicación. Los resultados mostrados en este trabajo están relacionado con el stellarator heliac TJ-II, pero el haber empleado esta metodología permite que la aplicación también sea empleada con otros dispositivos tales como LHD o ITER.

C.2. Arquitectura del *workflow*

A la hora de diseñar el *workflow* FastDEP se tuvieron en cuenta varios casos de uso, que dieron como resultado una herramienta flexible y adaptable a las necesidades del usuario. La figura C.1 muestra de manera gráfica el flujo del *workflow* FastDEP según dichos casos de uso.

C.2.1. Flujo de datos

La primera columna muestra el diseño original del *workflow*. En una primera instancia se ejecuta FAFNER2 en Grid empleando Montera. Los resultados se recogen en el nodo local, y cuando todas las ejecuciones han concluido se ejecutan los *wrapper* de post-proceso para obtener el resultado final. A continuación ISDEP es también ejecutado usando dicho resultado como datos de entrada.

Este proceso fue automatizado de dos maneras diferentes, una en la que el usuario verifica los resultados de FAFNER2 y otra en la que todo se ejecuta de manera automática.

Hay que tener en cuenta que, a pesar de que FAFNER2 e ISDEP sean códigos acoplables, no todas las ejecuciones de FAFNER2 proporcionan datos útiles para ISDEP. Por ejemplo, puede imaginarse una simulación donde según los parámetros indicados no hay partículas neutras que se ionicen en el plasma: ISDEP no tendría nada que simular, y su ejecución supondría una pérdida de tiempo y recursos computacionales. Así pues, se decidió la creación de un *workflow* en el que el usuario es capaz de verificar la salida de FAFNER2, decidiendo si es conveniente la ejecución de ISDEP con esos datos.

La solución propuesta pasa por el empleo de un servidor web para notificar al usuario el resultado de su simulación. Cuando la ejecución de FAFNER2 ha concluido y los *wrapper* de post-proceso han obtenido los resultados finales, la posición de las partículas es representada en una serie de

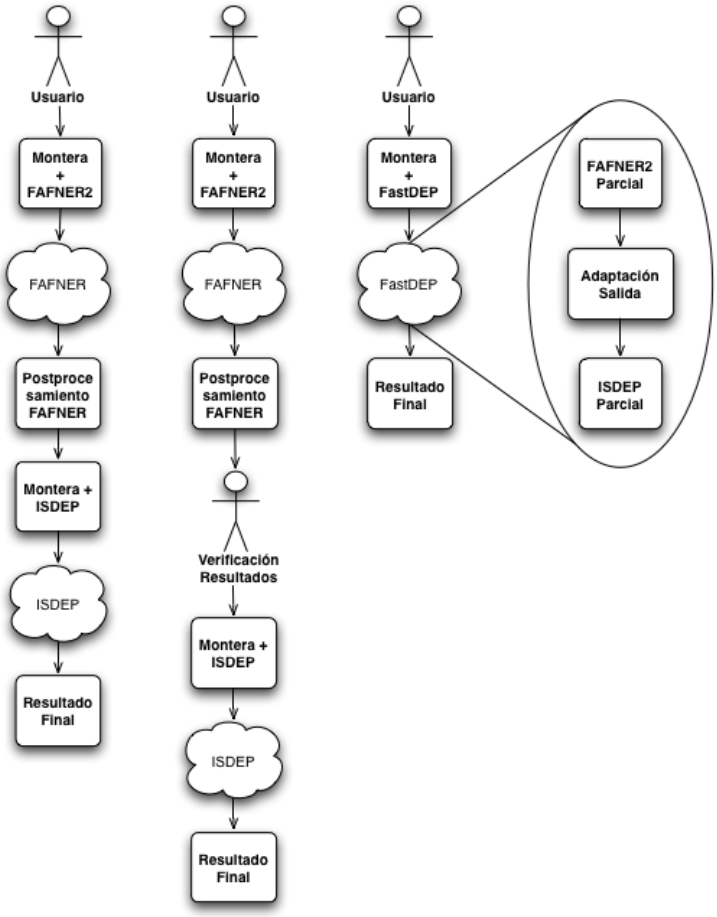

Figura C.1: Flujo de la aplicación FastDEP en sus distintas versiones.

gráficas tridimensionales empleando GNUPlot y volcada a un fichero *postscript*. Este fichero es enviado al usuario por correo electrónico junto a dos enlaces de hipertexto: el primero permite al usuario mostrar su satisfacción con el resultado de FAFNER2 y su deseo de que ISDEP se ejecute; el segundo, por el contrario, anula la ejecución de ISDEP y elimina los datos de la ejecución FAFNER2. Estos enlaces apuntan a páginas en PHP del servidor presente en el *host* donde se ejecuta Montera, las cuales ejecutan *scripts* que se encargan de llevar a cabo la operación solicitada.

La tercera implementación del *workflow* es sin duda la más agresiva, que permite un mayor rendimiento y muestra toda la potencia y flexibilidad de los códigos Monte Carlo aplicados a aplicaciones distribuidas.

Como se ha descrito en los apéndices anteriores, FAFNER2 e ISDEP tienen un funcionamiento similar en sus versiones distribuidas. Una aplicación

local decide el número de partículas a simular en cada *site* remoto, se envía un ejecutable junto con los datos de entrada, se lleva a cabo la ejecución y se devuelven los datos de salida que son posteriormente procesados para obtener el resultado final.

Si lo que se desea es acoplar FAFNER2 a ISDEP el proceso pasa porque cada instancia de ISDEP ejecute un subconjunto de los datos sobre iones simulados por FAFNER2. Por supuesto, si se emplean el mismo número de instancias de FAFNER2 e ISDEP, el subconjunto empleado por la segunda puede ser la salida de la primera, empleando para unirlos una pequeña aplicación que se encarga de formatear los datos de manera que no haya problemas de formato -básicamente parsea el XML devuelto por FAFNER2 y lo convierte en un archivo binario que ISDEP es capaz de leer. Así pues, en cada instancia de FastDEP son enviados al Grid tres ejecutables: FAFNER2, ISDEP y el código de acoplamiento. Los datos de entrada de FastDEP son los que corresponden a la instancia de FAFNER2, y los de salida son los de ISDEP.

Este planteamiento permite reducir en gran medida los *overheads* producidos por la ejecución distribuida de tareas. Por un lado los tiempos relacionados con la ejecución en Grid (tiempo de *scheduling*, colas, etc.) han de ser esperados una única vez por instancia de FastDEP, lo que los reduce en un 50 %. Por otro lado se evita el cuello de botella de la unión de los datos de cada instancia de FAFNER2 para obtener el resultado final, resultado que por otra parte no es de interés en este caso.

Por último, es importante recordar que en todos los casos es necesario ejecutar la aplicación que post-procesa los resultados parciales de ISDEP después de su ejecución en Grid. Esta aplicación fue modificada en el proceso de adaptación de ISDEP a Montera, tal y como ha sido descrito en el apéndice B.2.1.

C.3. Conclusiones

La dinámica de iones inyectados por NBI en diferentes configuraciones magnéticas tridimensionales pueden ser estudiadas de manera automática empleando FastDEP. Este *workflow* está basado en tecnologías Grid, aprovechandose de la naturaleza distribuida de los dos códigos Monte Carlo involucrados en el cálculo.

Un uso intensivo de este *workflow* permitirá estudiar las dinámicas de iones rápidos NBI en el ITER, siendo un punto clave para evaluar las propiedades del confinamiento de ese dispositivo. En un futuro esta distribución será tomada como una fuente que modificará el plasma de fondo, lo que permitirá estimar el calentamiento y momento de una manera autoconsistente, de una manera similar a la manera en la que la evolución del plasma de fondo fue estimada en (151).

Las pruebas muestran que la aproximación propuesta de FastDEP es computacionalmente eficiente y útil desde un punto de vista físico, por lo que su implementación puede considerarse un éxito.

Apéndice D

Publicaciones a las que ha dado lugar

D.1. Compendio

D.1.1. Incluidas en JCR

M. Rodríguez-Pascual, I.M. Llorente, R. Mayo. *Montera: A Framework for the Efficient Execution of Monte Carlo Codes on Grid Infrastructures*. Computing and Informatics (2011), *in print*. Factor de impacto: 0.356 (Q4)

M. Rodríguez-Pascual, J. Guasp, F. Castejón, A.J. Rubio-Montero, I.M. Llorente, R. Mayo. *Improvements on the Fusion Code FAFNER2*. IEEE Transactions on Plasma Science (2010) vol. 38 (9), pp 2102-2110. Factor de impacto: 1.076 (Q3)

D.1.2. Incluidas en CORE

M. Rodríguez-Pascual, A. J. Rubio-Montero, R. Mayo, A. Bustos, F. Castejón, I.M. Llorente. *More Efficient Executions of Monte Carlo Fusion Codes by Means of Montera: The ISDEP Use Case*. 19th Euromicro Conference on Parallel, Distributed and Network-based Processing (2011) vol. 1, pp. 380-384. Core: C

M. Rodríguez-Pascual, F. Castejón, A.J. Rubio-Montero, I.M Llorente, R. Mayo. *FAFNER2: A Comparison between the Grid and the MPI Versions of the Code*. 2010 International Conference on High Performance Computing & Simulations (2010) vol.1 pp 78-84. Core: B

M. Rodríguez-Pascual, J.Guasp, F.Castejón, A.J. Rubio-Montero, R. Mayo. *A Grid version of the Fusion code FAFNER*. 18th Euromicro Conference on Parallel, Distributed and Network-based Processing (2010), pp.449 - 453. Core: C

D.1.3. Capítulos de libro

M. Rodríguez-Pascual, J. Guasp, F. Castejón, I.M. Llorente, R. Mayo. *FAFNER2: Executions of a code for estimating NBI heating of fusion plasmas on Grid*. Proceeding of the Second EELA-2 Conference (2009), ISBN: 978-84-7834-627-1, pp 117-124.

M. Rodríguez-Pascual, J.Guasp, F.Castejón, A.J. Rubio-Montero, R. Mayo. *FAFNER2: adaptation of a code for estimating NBI heating of fusion plasmas on the Grid*. Proceedings of the First EELA-2 Conference (2009), ISBN: 978-84-7834-600-4, pp. 227-234.

D.1.4. Contribuciones a congresos

D.1.4.1. Charlas invitadas

M. Rodríguez-Pascual, *Montera: Un framework para la ejecución eficiente de códigos Monte Carlo*. Workshop en HPC del Centro de Modelamiento Matemático (2010). Santiago, Chile.

M. Rodríguez-Pascual, *DRMAA para aplicaciones científicas*. Workshop en HPC del Centro de Modelamiento Matemático (2010). Santiago, Chile.

D.1.4.2. Presentaciones

M. Rodríguez-Pascual, A. J. Rubio-Montero, R. Mayo, A. Bustos, F. Castejón. *More Efficient Executions of Monte Carlo Fusion Codes by Means of Montera: The ISDEP Use Case*. 19th Euromicro Conference on Parallel, Distributed and Network-Based Processing (2011). Aya Napa, Chipre.

M. Rodríguez-Pascual, I.M. Llorente, R. Mayo. *Montera: a framework for efficient executions of Monte Carlo codes on the Grid*. 5TH EGEE User Forum (2010). Uppsala, Suecia.

M. Rodríguez-Pascual, F. Castejón, A.J. Rubio-Montero, I.M. Llorente, R. Mayo. *FAFNER2: A Comparison Between the Grid and the MPI Versions of The Code*. 2010 International Conference on High Performance Computing & Simulation (2010). Caen, Francia.

M. Rodríguez-Pascual J.Guasp, F.Castejón, I.M.Llorente, R. Mayo. *Simulation of NBI Heating on Grid*. XXXII Reunión Bienal de la Real Sociedad Española de Física (2009). Ciudad Real, España.

D.1.4.3. Pósters

M. Rodríguez-Pascual J.Guasp, F.Castejón, A.J. Rubio-Montero, R. Mayo. *NBI heating simulations of fusion plasmas on the Grid (FAFNER2)*. 21th International Conference on Numerical Simulation of Plasmas (2009). Lisboa, Portugal.

M. Rodríguez-Pascual J.Guasp, F.Castejón, I.M.Llorente, A.J. Rubio-Montero, R. Mayo. *Improvements on the EGEE fusion code FAFNER-2*. 4th EGEE User Forum (2009). Catania, Italia.

M. Rodríguez-Pascual J.Guasp, F.Castejón, A.J. Rubio-Montero, R. Mayo. *FAFNER-2: adaptation of a code for estimating NBI heating of fusion plasmas on the Grid*. EGEE 2008 conference (2008). Estambul, Turquía.

D.2. Montera: A Framework for the Efficient Execution of Monte Carlo Codes on Grid Infrastructures

D.2.1. Cita completa

Manuel Rodríguez-Pascual, Ignacio Martín Llorente, Rafael Mayo. *Montera: A Framework for the Efficient Execution of Monte Carlo Codes on Grid Infrastructures*. Computing and Informatics (2011), *in print*.

D.2.2. Información sobre la revista

Factor de impacto: 0.356 Ranking: Q4

D.2.3. Abstract

The objective of this work is to improve the performance of Monte Carlo codes on Grid production infrastructures. To do so, the codes and the Grid *sites* are characterized with simple parameters to model their behaviors. Then, a new performance model for Grid infrastructures is proposed, and an algorithm that employs this information is described. This algorithm dynamically calculates the number and size of tasks to execute on each *site* to maximize the performance and reduce *makespan*. Finally, a newly developed framework called Montera is presented. Montera deals with the execution of Monte Carlo codes in an unattended way, isolating the complexity of the problem from the final user. By employing two fusion Monte Carlo codes as example cases, along with the described characterizations and scheduling algorithm, a performance improvement up to 650 % over current best results is obtained on a real production infrastructure, together with enhanced stability and robustness.

D.2.4. Referencia de Citas Bibliográficas

Ganglia Web, Nagios Web, I. Foster (2002), P. Andreo (1991), M. Arellano *et al* (1991), H. Biesel (1977), P. P. Boyle *et al* (1998), F. Castejón *et al* (2007), A. T. Chronopoulos *et al* (2001), H. J. Curnow *et al* (1976), R. Diacovo (2010), J. Díaz *et al* (2009), F. Dong *et al* (2006), R. Eckhard *et al* (1987), Y.-W. Fann *et al* (2000), J. Hernández *et al* (2008), J. Herrer (2009), J. Herrera *et al* (2006), D. B. Hertz (1964), R. W. Hockney *et al* (1988), E. Huedo *et al* (2005), E. Huedo *et al* (2006), S.-H. Jang *et al* (2004), G. Kitagawa (1996), C. Li *et al* (2007), Y. Li *et al* (2002), G. Lister (1985), L. Maigne *et al* (2004), L. Malcolm *et al* (1993), M. Mascagni *et al* (2003), N. Metropolis (1987), R. Montero *et al* (2006), H. Renshall (2004), M. Rodríguez-Pascual *et al* (2010), R. Roveda *et al* (2000), J. M. Schopf

et al (2005), D. Silva *et al* (2004), A. Teubel *et al* (1994), P. Troger *et al* (2007), T. H. Tzen *et al* (1993), J. L. Vázquez-Poletti *et al* (2007), J. L. Vazquez-Poletti *et al* (2008), M. Wu *et al* (2006), C.-T. Yang *et al* (2003), C. Yu *et al* (2010).

D.3. Improvements on the Fusion Code FAFNER2

D.3.1. Cita completa

Manuel Rodríguez-Pascual, José Guasp, Francisco Castejón, Antonio Juan Rubio-Montero, Ignacio Martín Llorente, Rafael Mayo. *Improvements on the Fusion Code FAFNER2*. IEEE Transactions on Plasma Science (Sept. 2010) vol. 38, issue 9, pp 2102-2110.

D.3.2. Información sobre la revista

Factor de impacto: 1.076 Ranking: Q3

D.3.3. Abstract

FAFNER2, a 3D code adapted to the TJ-II helical axis stellarator from the original one developed by Lister at Max Planck IPP, simulates by Monte Carlo methods the Neutral Beam Injection (NBI) technology (a key heating method for most of the fusion experiments worldwide).

To the date, FAFNER2 has been usually run at CIEMAT by means of a batch mode on a shared memory computer with MIPS processors. This work describes how the application has ben updated to employ MPI and run on standard Linux clusters. As in Grid infrastructures not every site has MPI installed on their nodes, a serial version has also ben developed, together with a Java DRMAA program able to run FAFNER2 on distributed resource platforms, such as Grid infrastructures. In addition, and with new improvements in the code for maximizing its performance, the metascheduler GridWay has been also incor- porated for maximizing the executions on the Grid. Scalability, fault tolerance and correctness of the result have been proved employing a significant number of particles and nodes within a local cluster and the EGEE infrastructure.

D.3.4. Referencia de Citas Bibliográficas

N. Miyamoto *et al* (1996), A. Rubio-Montero *et al* (2010), G. Lister *et al* (1985), S. Sato *et al* (2005), E. R. Hodgson *et al* (1998.), H. Renshall (2004), A.Teubel *et al* (1994), F. Castejón *et al* (2007), K. Feind *et al* (1995), A. Geist *et al* (1996), P. Troger *et al* (2007), E. Huedo *et al* (2006), M. Bessenrodt-Weberpals *et al* (1980), A. Teubel *et al* (1994), J. Guasp *et al* (1999), J. Guasp *et al* (2000), I. Foster (2002), F. Berman *et al* (2005), P.Herrero *et al* (1994), E. Huedo *et al* (2003), V. I. Vargas *et al* (2009), F. Castejón *et al* (2009).

D.4. More Efficient Executions of Monte Carlo Fusion Codes by Means of Montera: The ISDEP Use Case

D.4.1. Cita completa

Manuel Rodríguez-Pascual, Antonio Juan Rubio-Montero, Rafael Mayo, Andrés Bustos, Francisco Castejón, Ignacio Martín Llorente. *More Efficient Executions of Monte Carlo Fusion Codes by Means of Montera: The ISDEP Use Case*. 19th Euromicro Conference on Parallel, Distributed and Network-based Processing (Feb. 2011), vol. 1, pp. 380-384.

D.4.2. Información sobre la conferencia

Core: C

D.4.3. Abstract

ISDEP (Integrator of Stochastic Differential Equations for Plasmas) is a Monte Carlo code that solves the plasma dynamics in a fusion device and perfectly scales on distributed computing platforms. Montera is a recent framework developed for achieving Grid efficient executions of Monte Carlo applications, as ISDEP is. In this work, the improvement of performing the calculations of ISDEP with Montera, which rise up to 34.9 %, is shown as well as an analysis on the implications it could have, which aim to show to the fusion research community the benefits of using Montera.

D.4.4. Referencia de Citas Bibliográficas

H. Wobig (1999), J. P. Christiansen *et al* (2004), F.Castejón *et al* (2007), C. Alejaldre *et al* (1990), J. L. Velasco *et al* (2008), M. Rodríguez-Pascual *et al* (2011), R. J. Goldston *et al* (1995), T. S. Chen (1988), H. Curnow (1976), R. W. Hockney *et al* (1988), R. S. Montero *et al* (2006), J. Herrera *et al* (2008), E. Huedo *et al* (2005), J. Vázquez-Poletti *et al* (2007), L. Maigne *et al* (2004), H. Tanizaki (2000).

D.5. FAFNER2: A Comparison between the Grid and the MPI Versions of the Code

D.5.1. Cita completa

Manuel Rodríguez-Pascual, Francisco Castejón, Antonio Juan Rubio-Montero, Rafael Mayo. *FAFNER2: A Comparison between the Grid and the MPI Versions of the Code*. 2010 International Conference on High Performance Computing & Simulations (2010) vol.1 pp 78-84

D.5.2. Información sobre la conferencia

Core: B

D.5.3. Abstract

FAFNER2 is a 3D code that simulates by Monte Carlo methods the Neutral Beam Injection (NBI) technology. The original version was implemented for shared memory computers with MIPS processors, so an update to be executed by means of MPI on standard Linux clusters has been carried out as well as a new version to be run on Grid. To do the latest, a serial version has also been developed, together with a Java DRMAA program that is submitted by the GridWay metascheduler. As a result, two new improved versions of the code (HPC and Grid) are available.

D.5.4. Referencia de Citas Bibliográficas

N. Miyamoto *et al* (1996), G. Lister (1985), S. Sato *et al* (2005), A.Teubel *et al* (1994), A. H. Peter Troger *et al* (2007), E. Huedo *et al* (2006), A. Boozer (1980), A. Boozer (1983), A. Teubel *et al* (1994), J. Guasp *et al* (1999), J. Guasp *et al* (2000), M. Gobbin *et al* (2009), I. Foster (2002), I. Foster (2006), J.Heikkinen *et al* (1993), H. Mazzocco *et al* (2008), F. Castejón *et al* (2007), T. Desell *et al* (2007).

D.6. A Grid version of the Fusion code FAFNER

D.6.1. Cita completa

Manuel Rodríguez-Pascual, José Guasp, Francisco Castejón, Antonio Juan Rubio-Montero, Rafael Mayo. *A Grid version of the Fusion code FAFNER.* 18th Euromicro Conference on Parallel, Distributed and Network-based Processing (2010), pp.449 - 453.

D.6.2. Información sobre la conferencia

Core: C

D.6.3. Abstract

FAFNER is code developed by Lister which simulates by Monte Carlo methods the Neutral Beam Injection (NBI) technology, one of the most extended heating methods for fusion devices. To the date, FAFNER has been usually run at CIEMAT adapted to the TJ-II helical axis stellarator on shared memory Cray architecture machines. From this version, FAFNER has been ported to the Grid in the framework of the EGEE Project. At the same time, this work is the first step of a more ambitious work since the code can be now coupled to many others such as ion transport tools. In this paper, all these preliminary advances are described as well as the performance and portability gains obtained

D.6.4. Referencia de Citas Bibliográficas

T. Oikawa *et al* (2007), G. Lister (1985), J. A. Teubel *et al* (1994), E. Huedo *et al* (2005), E. Huedo *et al* (2006), G. R. Luecke *et al* (2004), J. L. Velasco *et al* (2008).

D.7. FAFNER2: Executions of a code for estimating NBI heating of fusion plasmas on Grid

D.7.1. Cita completa

Manuel Rodríguez-Pascual, José Guasp, Francisco Castejón, Ignacio Martín Llorente, Rafael Mayo. *FAFNER2: Executions of a code for estimating NBI heating of fusion plasmas on Grid*. Proceeding of the Second EELA-2 Conference (2009), ISBN: 978-84-7834-627-1, pp 117-124.

D.7.2. Abstract

FAFNER2 is a 3D code that simulates by Monte Carlo techniques the Neutral Beam Injection (NBI) technology, one of the most used heating method for fusion devices. It is adapted to the TJ-II helical axis stellarator from the original one developed by Lister at Max Planck IPP, but can easily be executed for other geometries by changing the independent associated module. In a first step, FAFNER2 was adapted to be run on the current architectures and paradigms (x86, MPI & DRMAA) also improving the old efficiency on shared memory Cray architectures. Nowadays, it can be executed on Grid and supercomputing platforms, but a real test on a production infrastructure was remaining. This work intends to overcome this issue by performing a massive test, the computational results of which will be a key factor for the final users in order to plan their calculus in a two-fold way: time spent and required accuracy in the physical values obtained.

D.7.3. Referencia de Citas Bibliográficas

G. Lister (1985), J.A. Teubel (1994), M. Rodriguez-Pascual *et al* (2009), E. Huedo *et al* (2005)

D.8. FAFNER2: adaptation of a code for estimating NBI heating of fusion plasmas on the Grid

D.8.1. Cita completa

Manuel Rodríguez-Pascual, José Guasp, F.Castejón, Antonio Juan Rubio-Montero, Rafael Mayo. *FAFNER2: adaptation of a code for estimating NBI heating of fusion plasmas on the Grid*. Proceedings of the First EELA-2 Conference (2009), ISBN: 978-84-7834-600-4, pp. 227-234.

D.8.2. Abstract

FAFNER2, a 3D code adapted to the TJ-II helical axis stellarator from the original one developed by Lister at Max Planck IPP, simulates by Monte Carlo methods the Neutral Beam Injection (NBI) technology (a key heating method for most of the fusion experiments worldwide).

To the date, FAFNER2 has been usually run at CIEMAT by means of a batch mode on a shared memory of a Cray architecture, but it has been also translated to MPI. From this point of view, FAFNER2 would not only mean a new code ported to the Grid, but it opens a new strategy for the fusion community. Since it deals with the heating method for the fusion devices, it is able to be coupled to ion kinetic transport codes in a way that the outputs from FAFNER2 will be the input for the latter. Such a complex workflow, the development of which could be done by means of Kepler Project tools, is of interest for both fusion and grid communities. Into this framework, the FAFNER2 porting process to the grid is the first step of a more ambitious work.

In this paper, all this preliminary advances will be described. The steps of the transformation from a SHMEM over MIPS to a Grid over a X86 technology will be fully explained. Also, performance and portability gains obtained on each step of the process will be evaluated.

D.8.3. Referencia de Citas Bibliográficas

A.Teubel (1994), E. Huedo *et al* (2006), G. Lister (1985).

Bibliografía

[1] Foster, I.: The anatomy of the grid: Enabling scalable virtual organizations. **15**(3) (2001) 200

[2] Foster, I.: What is the Grid? A Three Point Checklist. Grid Today **1** (2002)

[3] Iosup, A., et al.: The Grid Workoad Archive. Future Generation Computer Systems **24**(7) (2008) 672–686

[4] Sun Grid Engine Home: http://gridengine.sunsource.net

[5] PBS Home: http://www.pbsworks.com

[6] Korpela, E., Werthimer, D., Anderson, D.R., Cobb, J., Leboisky, M.: SETI@ home-massively distributed computing for seti. Computing in Science & Engineering **3**(1) (2002) 78–83

[7] Benito, D., et al.: Ibercivis: infrastructure, applications and development workflow. 3rd Iberian Grid Infrastructure Conference Proceedings) (May 2009) 3–8

[8] Anderson, D.R.: BOINC: A System for Public-Resource Computing and Storage. (2004) 4–10

[9] Huedo, E., Montero, R.S., Llorente, I.M.: The GridWay framework for adaptive scheduling and execution on grids. Scalable Computing-Practice and Experience **6**(8) (2005) 1–8

[10] Globus Toolkit 4 Home: http://www.globus.org

[11] Object Home: http://consortium.objectweb.org

[12] Sotomayor, B., Childers, L.: Globus Toolkit 4: Programming Java Services. (2006)

[13] Laure, E., et al.: Middleware for the next generation Grid infrastructure. (EGEE-PUB-2004-002) (2004) 4 p

[14] EGEE Home: http://public.eu-egee.org

[15] Brumfiel, G.: High-energy physics: Down the petabyte highway. Nature **469**(7330) (January 2011) 282–283

[16] Roy, A., Consortium, T.O.: Building and testing a production quality grid software distribution for the Open Science Grid. Journal of Physics: Conference Series **180** (August 2009) 012052

[17] EGEE Porting Projects Home: http://cagraidsvr06.cs.tcd.ie/porting

[18] Frey, J., Tannenbaum, T., Livny, M., Tuecke, S.: Condor-G: A Computation Management Agent for Multi-Institutional Grids. Cluster Computing **5** (2002) 237–246 10.1023/A:1015617019423.

[19] gLite Home: http://glite.web.cern.ch

[20] Andreetto, P., et al.: The gLite workload management system. Journal of Physics: Conference Series **119**(6) (2008) 062007

[21] Aiftimiei, C., et al.: Design and implementation of the gLite CREAM job management service. Future Generation Computer Systems **26**(4) (2010) 654–667

[22] Bell, M.: M. Bell's Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley (2008)

[23] Ferguson, D.F., Foster, I., Frey, J., Graham, S., Seukhin, I., Snelling, D., Tuecke, S., Vambenepe, W.: The ws-resource framework (2004)

[24] Foster, I., et al.: The Open Grid Services Architecture. Technical report, Global Grid Forum (GGF) (January 2005)

[25] Global Grid Forum Home: http://www.gridforum.org

[26] Aiftimiei, C., et al.: Using CREAM and CEMonitor for job submission and management in the gLite middleware. Journal of Physics: Conference Series **219**(6) (2010) 062001–062008

[27] Meglio, A., Bégin, M., Couvares, P., Takacs, E.: ETICS: the international software engineering service for the grid. Journal of Physics: Conference Series **119** (2008) 042010

[28] EGI Home: http://web.eu-egi.eu

[29] Datos EGI Mayo 2010: https://documents.egi.eu/public/ShowDocument?docid=42. EGI Document Server (May 2010)

[30] DTeam VO Home: https://wiki.egi.eu/wiki/Dteam_vo

[31] EGEE Fusion VO Home: http://grid.bifi.unizar.es/egee/fusion-vo

[32] Caron, E., Garonne, V., Tsaregorodtsev, A.: Evaluation of meta-scheduler architectures and task assignment policies for high throughput computing. The 4th International Symposium on Parallel and Distributed Computing, University of Lille, France (2005)

[33] Huedo, E., Montero, R.S., Llorente, I.M.: A recursive architecture for hierarchical grid resource management. Future Generation Computer Systems **25**(4) (2009) 401–405

[34] Andreetto, P., Dorigo, A., Gianelle, A., Zangrando, L., di Giusto, D.: Practical approaches to grid workload and resource management in the EGEE project. Proceedings of the International Conference on Computing in High Energy Physics (CHEP2004), Interlaken, Switzerland (2004)

[35] Huedo, E., Montero, R.S., Llorente, I.M.: Evaluating the reliability of computational grids from the end user's point of view. Journal of Systems Architecture **52**(12) (2006) 727–736

[36] Montero, R.S., Llorente, I.M.: A framework for adaptive execution in grids. Software: Practice and Experience **34**(7) (May 2004) 631–651

[37] Bosa, K., Schreiner, W.: Autrian Grid: Report on Experiments with Globus 4 and gLite. Technical report, Research Institute for Symbolic Computation (RISC), Johanes Kepler University, Linz (October 2008)

[38] Vázquez-Poletti, J.L., Huedo, E., Montero, R.S., Llorente, I.M.: A comparison between two grid scheduling philosophies: EGEE WMS and Grid Way. Multiagent and Grid Systems **3**(4) (2007) 429–439

[39] Tröger, P., Domagalski, P.: Standardization of an API for Distributed Resource Management Systems. Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007) (May 2007) 619–626

[40] Würthwein, F.: Me-My friends-the grid. Technical report (2005)

[41] Baker, M., Buyya, R., Laforenza, D.: The Grid: International efforts in global computing. In: Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet. (2000)

[42] Herman-Izycka, U., Ejdys, M., Huguenin, K.: APIs and End Users. Cluster and Grid Computing (2006)

[43] Karonis, N., Toonen, B., Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. Journal of Parallel and Distributed Computing **63**(5) (2003) 551–563

[44] : Document for a standard message-passing interface. Technical report, University of Tenneessee (1994)

[45] MPich G2 Home: http://www3.niu.edu/mpi

[46] Smith, C., Computing, P.: A Simple API for Grid Applications (SAGA). OGF Report (2008)

[47] Seymour, K., YarKhan, A., Agrawal, S., Dongarra, J.J.: Netsolve: Grid enabling scientific computing environments. Advances in Parallel Computing **14** (2005) 33–51

[48] Agrawal, S., Dongarra, J.J.: Users'Guide to NetSolve V1. 4. (2001)

[49] Tanaka, Y., Nakada, H., Sekiguchi, S., Suzumura, T., Matsuoka, S.: Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing. Journal of Grid Computing **1** (2003) 41–51

[50] Seymour, K., Nakada, H., Matsuoka, S., Dongarra, J.J.: A GridRPC Model and API for End-User Applications. Institute of Electrical & Electronics Engineers(IEEE) (2005)

[51] Loureiro, M., Pan, M., Rodríguez-Pascual, M., Mayo García, R., Posada, D.: MPI and Grid selection models of nucleotide substitution. Iberian Grid Infrastructure Conf. Proc (4) (2010) 478–481

[52] Andreo, P.: Monte Carlo techniques in medical radiation physics. Physics in medicine and biology **36** (1991) 861

[53] Maigne, L., et al.: Parallelization of Monte Carlo simulations and submission to a Grid environment. Parallel processing letters **14**(2) (2004) 177–196

[54] Arellano, M., Bond, S.: Some tests of specification for panel data: Monte Carlo evidence and an application to employment equations. The Review of Economic Studies **58**(2) (1991) 277–297

[55] Tanizaki, H.: Nonlinear and non-Gaussian state-space modeling with Monte Carlo techniques: A survey and comparative study. Volume 21 of Handbook of Statistics. North-Holland (2001)

[56] Malcolm, L., Spaulding Eric, L., Anderson, Tatsu, I., Eoin, H.: Simulation of the oil trajectory and fate in the arabian gulf from the Mina Al Ahmadi spill. Marine Environmental Research **36**(2) (1993) 79–115

[57] Hernández, J., et al.: CMS Monte Carlo production in the WLCG computing grid. Journal of Physics: Conference Series **119** (2008) 052019–052028

[58] Teubel, A., Guasp, J., Liniers, M.: Monte Carlo simulations of NBI into the TJ-II Helical Axis Stellarator. Technical Report 4, Max-Planck-Institut für Plasmaphysik, Garching bei München (1994)

[59] Roveda, R., Goldstein, D.B., Varghese, P.L.: Hybrid Euler/Direct Simulation Monte Carlo Calculation of Unsteady Slit Flow. Journal of Spacecraft ad Rockets **37**(6) (2000)

[60] Konopinski, E., Marvin, C., Teller, E.: Ignition of the atmosphere with nuclear bombs. Los Alamos internal report (1946) 1–22

[61] Eckert Jr, J., Mauchly, J.: Electronic numerical integrator and computer. United States Patent Office (1964)

[62] Institute for Advanced Study Home: http://www.ias.edu

[63] Eckhard, R.: Stan Ulam, John Von Neumann and the Monte Carlo Method. Los Alamos Science **Special Issue** (1987)

[64] Metropolis, N.: The Beginning of the Monte Carlo Method. Los Alamos Science **Special Issue** (1987)

[65] Doolen, G., Hendricks, J.: Monte Carlo at work. Los Alamos Science, Special Issue Dedicated to S. Ulam (1987) 142–143

[66] Moore, G.E.: Cramming more components onto integrated circuits. Electronics **38**(8) (1965) 114–117

[67] Blog Corporativo de Intel: La Ley de Moore Continúa avanzando en Intel. (October 2009)

[68] Blog Corporativo de Intel: El fin de la era de los MHz y el inicio de la era Multi-Core. (July 2007)

[69] Top 500 Home: http://www.top500.org

[70] Pearson, K.: The problem of the random walk. Nature **72**(1865) (1905) 294

[71] Metropolis, N., et al.: Equation of state calculations by fast computing machines. The Journal of Chemical Physics **21** (1953) 1087–1092

[72] Diaconis, P.: The markov chain monte carlo revolution. AMERICAN MATHEMATICAL SOCIETY **46**(2) (2009) 179–205

[73] Hammersley, J.M., Handscomb, D.C.: Monte Carlo methods. Methuen's Monographs on Applied Probability & Statistics. John Wiley & Sons, Inc, New York (1964)

[74] Liu, J.S.: Monte Carlo Strategies in Scientific Computing. Springer Series in Statistics. Springer (2008)

[75] Hall, P.: Task Scheduling in Parallel and Distributed System. (1994)

[76] Casavant, T., Kuhl, J.: A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Transactions on Software Engineering **14**(2) (1988) 141–154

[77] Dong, F., Akl, S.G.: Scheduling algorithms for grid computing: State of the art and open problems. School of Computing (2006)

[78] Xhafa, F., Abraham, A.: Meta-heuristics for grid scheduling problems. Metaheuristics for Scheduling in Distributed Computing Environments (2008) 1–37

[79] Peris, A.D., Hernandez, J., Huedo, E., Llorente, I.M.: Data location-aware job scheduling in the grid. Application to the GridWay metascheduler. Journal of Physics: Conference Series **219** (2010) 062043

[80] Yu, C., Marinescu, D.C.: Algorithms for Divisible Load Scheduling of Data-intensive Applications. Journal of Grid Computing **8**(1) (March 2010) 133–155

[81] Deelman, E.: Grids and Clouds: Making Workflow Applications Work in Heterogeneous Distributed Environments. **24**(3) (August 2010) 284–298

[82] González-Vélez, H., Cole, M.: Adaptive structured parallelism for distributed heterogeneous architectures: a methodological approach with pipelines and farms. Concurrency and Computation: Practice and Experience **22** (October 2010) 2073–2094

[83] Nguyen, T.: An object-oriented model for adaptive high-performance computing on the computational GRID. ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE (2005) 142

[84] Sarkar, A., Roy, S., Ghosh, D., Mukhopadhyay, R., Mukherjee, N.: An Adaptive Execution Scheme for Achieving Guaranteed Performance in Computational Grids. Journal of Grid Computing **8**(1) (March 2010) 109–131

[85] Li, C., Li, L.: Utility-based QoS optimisation strategy for multi-criteria scheduling on the grid. Journal of Parallel and Distributed Computing **67**(2) (2007) 142–153

[86] Li, Y., Mascagni, M.: Grid-based Monte Carlo Application. Grid (2002) 13–24

[87] Mascagni, M., Li, Y.: Computational Infrastructure for Parallel, Distributed, and Grid-based Monte Carlo Computations. Lecture Notes in Computer Sciences **2907** (2003) 39–52

[88] Silva, D., Cirne, W., Brasileiro, F.: Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. Euro-Par 2003 Parallel Processing (2004) 169–180

[89] Vazquez-Poletti, J.L., Huedo, E., Montero, R.S., Llorente, I.M.: CD-HIT Workflow Execution on Grids Using Replication Heuristics. 8th IEEE International Symposium on Cluster Computing and the Grid (2008) 735–740

[90] Wild, P., Johnson, P., Johnson, H.: Understanding task grouping strategies. Proceedings of HCI 2003: Designing for Society (2003) 3–20

[91] Chronopoulos, A.T., Benche, M., Grosu, D., Andonie, R.: A class of loop self-scheduling for heterogeneous clusters. Proceedings of the 3rd IEEE International Conference on Cluster Computing (2001) 282–291

[92] Herrera Sanz, J., Huedo, E., Montero, R.S., Llorente, I.M.: Loosely-coupled loop scheduling in computational grids. 20th Internationa Parallel and Distributed Processing Symposium (2006) 6

[93] Herrea Sanz, J.: Modelo de programación para infraestructuras Grid computacionales. PhD thesis, Universidad Complutense de Madrid, Madrid (2009)

[94] Yang, C.T., Chang, S.C.: A parallel loop self-scheduling on extremely heterogeneous PC clusters. Proceedings of Intl. Conference on Computational Science (2003) 1079–1088

[95] Fann, Y.W., Yang, C.T., Tseng, S.S., Tsay, C.J.: An intelligent parallel loop scheduling for parallelizing compilers. Journal of Informatic Science and Engineering (16) (2000) 169–200

[96] Jang, S.H., Wu, X., Taylor, V.: Using performance prediction to allocate grid resources. Technical report, GriPhyN Technical Report (2004)

[97] Díaz, J., Reyes, S., Niño, A., Muñoz-Caro, C.: Derivation of self-scheduling algorithms for heterogeneous distributed computer systems: Application to internet-based grids of computers. Future Generation Computer Systems **25**(6) (April 2009) 617–626

[98] Wu, M., Sun, X.: Grid harvest service: a performance system of grid computing. Journal of Parallel and Distributed Computing **66**(10) (2006) 1322–1337

[99] Bierwirth, C., Mattfeld, D.: Production scheduling and rescheduling with genetic algorithms. Evolutionary computation **7**(1) (1999) 1–17

[100] Gao, Y., Rong, H., Huang, J.: Adaptive grid job scheduling with genetic algorithms. Future Generation Computer Systems **21**(1) (2005) 151–161

[101] Li, Y., Yang, Y., Ma, M., Zhou, L.: A hybrid load balancing strategy of sequential tasks for grid computing environments. Future Generation Computer Systems **25**(8) (May 2009) 819–828

[102] Shin, W., Yang, C.T., Tseng, S.S.: Using a performance-based skeleton to implement divisible load applications on grid computing environments. Journal of Information Science and Engineering **25**(1) (2009) 59–81

[103] Ramírez-Alcaraz, J., et al.: Job Allocation Strategies with User Run Time Estimates for Online Scheduling in Hierarchical Grids. Journal of Grid Computing (January 2011) 95–116

[104] Dijkstra, E.W.: A case against the GO TO statement. Communications ACM (1968)

[105] Lilja, D.J.: Exploiting the parallelism available in loops. COMPUTER, **27** (1994) 13–26

[106] Fang, Z., Tang, P., Yew, P.C., Zhu, C.Q.: Dynamic Processor Self-Scheduling for General Parallel Nested Loops. IEEE Transactions on Computers **39** (July 1990) 919–929

[107] Polychronopoulos, C.D., Kuck, D.J.: Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. IEEE Transactions on Computers **36** (December 1987) 1425–1439

[108] Tzen, T.H., Ni, L.M.: Trapezoid Self-Scheduling: A Practical Scheduling Scheme for Parallel Compilers. IEEE Transactions on Parallel and Distributed Systems **4** (1993) 87–98

[109] Hockney, R.W., Jesshope, C.R.: Parallel Computers Two: Architecture, Programming and Algorithms. (1988)

[110] Montero, R.S., Huedo, E., Llorente, I.M.: Benchmarking of high throughput computing applications on grids. Parallel Computing **32**(4) (2006) 267–279

[111] Curnow, H.J., Wichmann, B.A.: A syntetic Benchmark. Computer Journal **19**(1) (1976) 43–49

[112] Brooks, S.: Markov chain Monte Carlo method and its application. Journal of the Royal Statistical Society: Series D (The Statistician) **47**(1) (1998) 69–100

[113] Kitagawa, G.: Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. Journal of computational and graphical statistics **5**(1) (1996) 1–25

[114] Biesel, H.: Recursive calculation of the standard deviation with increased accuracy. Chromatographia **10** (1977) 173–175

[115] Rocha, L., Tarragó Pinto, M., Caliri, A.: The water factor in the protein-folding problem. Brazilian Journal of Physics **34** (March 2004) 90–101

[116] Czibula, G., Bocicor, M., Czibula, I.G.: Solving the Protein Folding Problem Using a Distributed Q-Learning Approach. International JOurnal of Computers **5**(3) (August 2011) 404–413

[117] Zhang, Y., Kihara, D., Skolnick, J.: Local energy landscape flattening: Parallel hyperbolic Monte Carlo sampling of protein folding. Proteins **48**(2) (June 2002) 192–201

[118] Bastolla, U., Frauenkron, H., Gerstner, E., Grassberger, P., Nadler, W.: Testing a new Monte Carlo algorithm for protein folding. Proteins **32**(1) (July 1998) 52–66

[119] Swendsen, R.H., Wang, J.S.: Replica Monte Carlo simulation of spin glasses. Physical Review Letters **57**(2) (November 1986) 2607–2609

[120] Lepage, G.P.: Lattice QCD for Novices. ArXiv High Energy Physics - Lattice e-prints (2005) 1–42

[121] Lukkarinen, J.: Lattice Simulations of the Quantum Microcanonical Ensemble. ArXiv High Energy Physics - Theory e-prints (1997) 1–12

[122] Jorgensen, W.L., Tirado Rives, J.: Molecular modeling of organic and biomolecular systems using BOSS and MCPRO. Journal of Computational Chemistry **26**(16) (2005) 1689–1700

[123] Ferguson, D., Siepmann, J.: Monte Carlo methods in chemical physics. Kluwer Academic Publishers (1999)

[124] Lister, G.: FAFNER: A Fully 3-D Neutral Beam Injection Code Using Monte Carlo Methods, Garching bei München (1985)

[125] Castejón Magaña, F., et al.: Ion kinetic transport in the presence of collisions and electric field in TJ-II ECRH plasmas. Plasma Physics and Controlled Fusion **49** (2007) 753–776

[126] Rodríguez-Pascual, M., et al.: Simulations of NBI Fast Ions Distribution in Stellerators Based on Coupled Monte Carlo Fueling and Orbit Monte Carlo Codes. Computer Physics Communications

[127] ITER Home: http://www.iter.org

[128] Rodríguez-Pascual, M., Tapiador, D., Fontán, J., Huedo, E., Montero, R.S., Llorente, I.M.: Dynamic provisioning of virtual clusters for grid computing. Euro-Par 2008 Workshops-Parallel Processing (2009) 23–32

[129] Kolmogoroff, A.: Uber die analytischen Methoden in der Wahrscheinlichkeitsrechnung. Mathematische Annalen **104**(1) (December 1931) 415–458 10.1007/BF01457949.

[130] Boozer, A.: Guiding center drift equations. Physics of Fluids **23** (1980) 904–909

[131] Langevin, P.: On the theory of Brownian motion. Comptes Rendus de lÁcadémie des Sciences, Paris (1908)

[132] de Bustos, A., et al.: Impact of 3D features on ion collisional transport in ITER. Nuclear Fusion **50**(12) (November 2010) 125007

[133] Rodríguez-Pascual, M., et al.: More Efficient Executions of Monte Carlo Fusion Codes by Means of Montera: The ISDEP Use Case. In: Proceedings of the 2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing, Washington, DC, USA, IEEE Computer Society (2011) 380–384

[134] Renshall, H.: New Time Units simplify procedures. CERN computer newsletter (November 2004)

[135] Lee, J., Ware, B.: Open Source Development with LAMP: Using Linux, Apache, MySQL, Perl, and PHP. 1 edn. Addison-Wesley Professional (December 2002)

[136] Motojima, O., et al.: Initial physics achievements of large helical device experiments. Physics of Plasmas **6**(5) (May 1999) 1843–1850

[137] Parker, C., Suleman, H.: A lightweight web interface to grid scheduling systems. Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology (2008) 180–187

[138] GridWay Ecosystem Home: http://www.gridway.org/doku.php?id=ecosystem:relatedsoftware

[139] Miyamoto, N., et al.: Experimental results on ITER-NBI concept source. AIP Conference Proceedings **380** (1996) 300–308

[140] Sato, S., HIida, Yamauchi, M., Nishitani, T.: Shielding design of the ITER NBI duct for nuclear and bremsstrahlung radiation. Radiation Protection Dosimetry **116**(1,4) (2005) 28–31

[141] Backus, J.: The history of FORTRAN I, II, and III. Annals of the History of Computing **1**(1) (1979) 165–180

[142] SGI Home: http://techpubs.sgi.com/library/tpl/cgi-bin/browse.cgi?coll=0650&db=man

[143] Forum, M.P.I.: MPI: A Message-Passing Interface Standard, Version 2.1. High Performance Computing Center Stuttgart (HLRS) (June 2008)

[144] NAG Home: http://www.nag.com

[145] Student: The probable error of a mean. Biometrika **6**(1) (1908) 1–25

[146] Christiansen, J.P., Connor, J.W.: Ion transport from collisions and finite guiding centre drift excursions. Plasma Physics and Controled Fusion **46** (2004) 1537–1566

[147] Wobig, H.: Theory of advanced stellarators. Plasma Physics and Controled Fusion **41** (1999) A159–A173

[148] Kurki-Suonio, T., et al.: Monte Carlo simulations of the heat load asymmetries on JET divertor plates. Nuclear Fusion **42** (2002) 725–732

[149] Shasharina, S.G., Cary, J.R.: Ripple induced stochastic loss of alpha particles in a tokamak. Nuclear Fusion (42) (2002) 725–732

[150] Alejaldre, C., et al.: TJ-II, a flexible heliac stellarator. Fusion Technology **17** (1990) 131–139

[151] Velasco, J.L., et al.: Ion heating in transitions to CERC in the stellarator TJ-II. Nuclear Fusion **48** (2008) 065008

[152] Goldston, R.J., Rutherford, P.H.: Introduction to Plasma Physics. Taylor & Francis (1995)

[153] Chen, T.S.: A general form of the coulomb scattering operators for Monte Carlo simulations and a note on the guiding center equations in different magnetic coordinate conventions. Technical Report 0/50, Max-Planck-Institut fuer Plasmaphysik, Garching bei München (1988)

[154] Gómez-Iglesias, A., Castejón Magaña, F.: Grid Computing for Fusion Research. In: Advances in Grid Computing. Intech. Rijeka, Rijeka, Croatia (February 2011) 215–228

[155] Murakami, S., et al.: A global simulation study of ICRF heating in the LHD. Nuclear Fusion **46**(7) (June 2006) S425–S432

[156] Terry, P.: Suppression of turbulence and transport by sheared flow. Reviews of Modern Physics **72** (2000) 109–165

[157] Rice, J.E., et al.: Inter-machine comparison of intrinsic toroidal rotation in tokamaks. Nuclear Fusion **47**(11) (October 2007) 1618–1624

[158] Helander, P., Simakov, A.: Intrinsic Ambipolarity and Rotation in Stellarators. Physical Review Letters **101**(14) (September 2008)

[159] Yoshinuma, M., Ida, K., Yokoyama, M., Nagaoka, K., Osakabe, M., the LHD Experimental Group: Observations of spontaneous toroidal flow in the LHD. Nuclear Fusion **49**(7) (July 2009) 075036

Computing and Informatics, Vol. 20, 2001, 1–32, V 2011-Aug-29

MONTERA: A FRAMEWORK FOR THE EFFICIENT EXECUTION OF MONTE CARLO CODES ON GRID INFRASTRUCTURES

Manuel Rodríguez-Pascual, Rafael Mayo

CIEMAT.
Avda Complutense, 22. 28040 Madrid (Spain)
e-mail: manuel.rodriguez@ciemat.es, rafael.mayo@ciemat.es

Ignacio M. Llorente

DSA-Research.org. Universidad Complutense.
C/ Prof. José García Santesmases s/n. 28040 Madrid (Spain)
e-mail: llorente@dacya.ucm.es

Abstract. The objective of this work is to improve the performance of Monte Carlo codes on Grid production infrastructures. To do so, the codes and the Grid *sites* are characterized with simple parameters to model their behaviors. Then, a new performance model for Grid infrastructures is proposed, and an algorithm that employs this information is described. This algorithm dynamically calculates the number and size of tasks to execute on each *site* to maximize the performance and reduce *makespan*. Finally, a newly developed framework called Montera is presented. Montera deals with the execution of Monte Carlo codes in an unattended way, isolating the complexity of the problem from the final user. By employing two fusion Monte Carlo codes as example cases, along with the described characterizations and scheduling algorithm, a performance improvement up to 650% over current best results is obtained on a real production infrastructure, together with enhanced stability and robustness.

Keywords: Scheduling; Task Grouping; Grid Computing; Monte Carlo; Perfor-

68W15 Distributed algorithms
68W40 Analysis of algorithms
65C05 Monte Carlo methods

mance

1 INTRODUCTION

Monte Carlo (MC) codes rely on the assumption that the behavior of complex phenomena can often be modeled by the execution of multiple simple simulations that employ random or pseudo-random numbers to compute their results. Thanks to this particularity, MC codes are widely used in various fields of computing simulations, solving problems where it is unfeasible or impossible to compute an exact result with a deterministic algorithm. The modularity of MC codes also simplifies the execution of distributed environments, where task distribution and synchronization is not trivial. MC codes have been successfully employed in fields as diverse as radiological medicine [4, 36], economics [5], finances [24, 8], environmental research [37], high energy physics [21, 49] or aerospace engineering [46], just to mention a few.

The kind of MC codes considered for this work are the ones consisting on sets of independent simulations, whose architecture allows an straightforward parallelization. Other kind of codes such as Random Walks [42] or Marcov's Chains [39, 15] are beyond the scope of this work. Thus, from now on, references to "Monte Carlo" should be understood as references to this subset of Monte Carlo-based applications.

Because of the high demand many scientific applications place on resources - MC codes among them- Grid computing has emerged as a powerful platform for facing new and more ambitious problems. Grid computing has enabled the scientific community to have easier access to large computing resources beyond supercomputers. The nature of MC codes makes their parallelization straightforward, and they have been successfully ported to the Grid on multiple occasions. However, the execution of MC codes still needs improvement to make the most of their flexibility in combination with the dynamicity of Grid infrastructures, the characteristics of which have either not been taken into account or have only been dealt with in ideal environments.

Due to the particularities of this kind of infrastructure, the efficient execution of distributed applications is far from trivial. As detailed in the Related Works section, enormous effort has been put into this area, with many different approaches focusing on different concepts of "Grid Infrastructure", the particular characteristics of the application to be executed or the available and published information. Still, there is no specific tool for executing MC based applications on the Grid, even though MC is one of the most widely employed paradigms. In this work, the problem is tackled through the creation of a framework specifically designed to fit the requirements of MC applications.

For this purpose, an innovative strategy that relies on three complementary tools is followed. These tools are employed to gather information from different sources and to distribute the samples to be simulated among the available resources. The three tools are as follows:

- information about the Grid infrastructure based on static and dynamic data collected from past executions and the current status;
- an automatic analysis and characterization of the application to execute to model its behavior; and,
- the employment of a new scheduling algorithm.

To carry on this task, a number of contributions are presented on this paper.

First, an innovative characterization of MC codes and Grid infrastructures with a small set of simple parameters is proposed. This way, the behavior of the application and infrastructure performance on any given moment can be accurately predicted, thus providing very valuable information to the scheduler.

The next step consisted on the design of a new scheduling algorithm called *Dynamic Trapezoidal Self Scheduling* (DyTSS). This algorithm is specially devoted to MC applications, and it employes the aforementioned characterizations to split the samples to simulate among the available resources. Unlike previous self-scheduling algorithms, which decide the task distribution and the *chunk* size at the beginning of the execution, DyTSS creates every *chunk* at the moment of its execution, thus adapting the submission of jobs to any change in the number, performance or size of the resources being used in the dynamic environment.

As manually controlling all this tools would be difficult and tedious for the final user, a new framework called Montera (*MONTE carlo RApido* - Fast Monte Carlo, from its Spanish acronym) is presented to overcome this issue. Montera carries out all the steps involved on a Grid execution of MC codes, from information retrieval and *site* characterization to advanced scheduling via DyTSS algorithm.

These tools will be thoroughly explained together with the most significant details of the implementation. The rest of the work is organized as follows: section 2 is devoted to show the related work on the area; section 3 explains the creation of Montera from a theoretical point of view and details the most significant characteristics of its implementation; in section 4 the performance an the induced overhead are analysed, showing the behavior of Montera on real production infrastructures; last, section 5 shows the conclusions and lessons learned during the development of this work.

2 RELATED WORKS

Job scheduling on Grid infrastructures is a widely studied field, and a complete description of the research in this area falls out of the scope of this work. If a general reference is desired, Dong and Akl [17] have performed an impressive analysis of the state-of-the-art scheduling algorithms for Grid computing, and a study on classical self-scheduling algorithms can be found in Chronopoulos' work [11] However, these approaches to the problem, although positive, do not offer a solution or a significant improvement to the execution of MC codes, as will be demonstrated in the following analysis. This achievable and necessary improvement is the aim and justification of the current work.

Note that the final objective is to reduce the *makespan* of a CPU-intensive application. Thus, techniques like those detailed in Li's work [32], which focus on QoS or Yu's work [57] which focuses on data intensive applications, are beyond the scope of this analysis.

It is important to regard that the definition of Grid Computing is far away from being achieved among the scientific community. In this case the definition established by Ian Foster [3] is followed. Here, a three-point checklist is proposed to determine whether a given system can be considered a Grid infrastructure or not. Regarding this list, it is a system that:

- Coordinates resources that are not subject to centralized control.
- Uses standard, open, general-purpose protocols and interfaces.
- Deliver nontrivial qualities of service.

The first point means that the resources can belong to different organizations, each one with different software, usage and security policies. This resource sharing must be highly controlled, so users, service providers and infrastructure administrators are aware of *when*, *to whom*, *for what* and *under which conditions* a certain resource is shared.

By requiring that the protocols and interfaces (*middleware*) are general-purpose the resulting infrastructure is not oriented to solve a certain kind of problem and/or application, but can be employed to different areas of knowledge. The employment of standard, open protocols encourages the users to adapt the tools to their specific needs, while avoiding fragmentation.

In addition, the resulting infrastructure must be reliable and stable enough to constitute a valid alternative to the users, so they can carry on their experiments.

Of course, the existence of Montera as an specific tool to execute MC codes does not collide with the second point of the list: that definition is devoted to the middleware, and allows building more sophisticates layers on top of it.

2.1 Task Replication

Given that each simulation by an MC code is fully independent, and all of them are equally valid for obtaining the final result, task replication represents a powerful tool for reducing execution time. With this approach, the code starts the execution of more tasks than would be strictly necessary. Then, when the desired number of simulations has been executed, the remaining tasks are aborted. With this approach, it is possible to avoid bottlenecks resulting from performance loss or the failure of a particular resource, thus minimizing *makespan*.

This technique has been previously studied within the framework of Grid computing [33, 38, 48, 53]. The results provided by the authors of each approach are highly variable, and they depend on the size of the problem, the number of tasks to execute and the status of the Grid infrastructure employed to make the measurements. It is important to bear in mind that Poletti's work [53] is the only one

performed in a real Grid infrastructure, whereas the rest only provide the results of a simulation.

2.2 Task Grouping

As previously mentioned, MC codes are constituted by fixed sections executed at the beginning and end of the execution and the simulation of an arbitrary number of samples. If the aforementioned Grid overheads are taken into account, and given that they are independent of the problem size, it makes sense to perform more than one simulation on each instance of the application.

Task grouping is a technique that has been widely applied in Grid scheduling [56, 19, 55, 28, 16, 22]. Here, several tasks are grouped into a single job called a *chunk*. This technique minimizes some of the overheads produced by the Grid execution model. The executable task is sent to the remote *site* only once, thus reducing transmission time, and the queue time has to be waited on only once per *chunk* instead of once per task. This approach is an adaptation of a technique for loop scheduling in distributed heterogeneous environments called "loop self-scheduling". Task grouping can be directly extrapolated to MC codes, considering that the *chunk* size corresponds to the number of samples executed in one task.

Nevertheless, in most of the current bibliography, key aspects related to a real dynamic Grid production infrastructure -dynamicity of computing resources, performance variations and high fault rate- have not been taken into account or the performed tests have been made in simulated, controlled environments.

3 MONTERA

In this section, the components and functionalities of Montera, the proposed framework for the efficient execution of MC codes on Grid infrastructures, will be described.

3.1 Characterization of Monte Carlo codes

To improve the execution of MC codes on the Grid, the first step of this work is to characterize them. The existence of a valid model allows the encapsulation of the application to be executed and its isolation from Montera, which can then seamlessly execute different codes.

Monte Carlo method obtains its name from the Principality of Monaco, "the gambling capital" [40]. It is based on the fact that the behavior of complex phenomena can be modeled by performing multiple independent, simple simulations, employing random or pseudo-random numbers to compute partial results and statistics to join them and obtain the problem solution [18].

MC applications -this is, applications based on MC method- are based on the simulation of an arbitrary number of independent samples, grouping them in one

or more tasks. A typical MC code is divided into three different sections. The first section performs the common operations at the beginning of the execution, such as checking input data and initializing data structures. The second section is the simulation of the desired number of samples. The third section joins the results of the previous simulations and analyzes them to obtain the final result. The execution time of each section depends on the number of samples to simulate, and the time of the first and last may also incorporate a constant time. Thus, the execution time of the entire code depends on the number of samples plus a constant time.

In this proposal, the MC codes are represented by two parameters: *constant_effort* and *sample_effort*. *constant_effort* represents the effort necessary to execute the constant part of the application, and *sample_effort* is the effort necessary to simulate a single sample.

MC codes are CPU-intensive [9, 30] and in many cases are devoted to floating-point operations. Therefore, the chosen unit for the proposed parameters is $seconds*whetstone$.

The Whetstone benchmark [12] is widely employed to estimate the speed of a computational resource when working with floating points. The unit in which the result is described is *Millions of Whetstone Instructions per Second* (MWIPS), which, for the sake of simplicity, will hereafter be shortened to whetstones. The whetstones of a *site* and the two mentioned parameters provide enough information to accurately estimate the execution time of a given instance of the code on a particular resource.

To obtain these values, a *code profiling* mechanism has been implemented in Montera. This mechanism submits to one or several *sites* a script that:

- Executes the Whetstone benchmark in the remote *site* to measure its performance.
- Executes the application to be profiled with an increasing number of samples, so *constant_effort* and *sample_effort* can be determined. This execution can last for a predetermined period or for the time the user considers necessary.

By repeating this process at different *sites*, the influence of exogenous factors, such as performance variations or benchmark imprecision, is minimized.

Finally, an analysis of the results based on the following equation is carried out.

$$Execution\ Time\ N\ Samples = Ce + N * Se \tag{1}$$

Where C_e corresponds to *constant_effort*, Se to *sample_effort* and N is the number of simulated samples in a given execution. As can be seen, obtaining C_e and S_e after two executions of the code with different number of samples is straightforward. However, to improve the accuracy of the results, this analysis is performed several times with an increasing value of N, and the parameters are obtained with a weighted average of the partial results. Finally, these values are normalized with the result of the benchmark execution.

3.2 Characterization of Grid sites

When dealing with task scheduling on Grid infrastructures, there are -as explained in section 2- two different approaches. The Grid infrastructure can be considered either a black box or a set of known resources. Here, the latter is employed, so a strong effort has been put toward gathering as much information as possible about the composition of the infrastructure.

To accomplish this task, the *sites* have been characterized according to the following parameters:

- Whetstones: efficiency when working with floating points.
- Queue time: average queue time for a task before being executed.
- Bandwidth.
- Number of successful and failed tasks in past executions.
- Available slots in past executions.
- Number of failed attempts to profile the *site* and when the last one occurred.

In the case of the first two parameters, both the average value and the typical deviations are stored.

With these parameters, Montera relies on more information about each *site* than previous tools, mainly based on its public information (e.g., architecture, CPU MHz). This approach represents a clear advantage for the task scheduling process, as a deeper knowledge of the Grid infrastructure leads to more accurate decisions.

Montera is able to profile the *sites* of the Grid infrastructure and to automatically obtain this information. When a new *site* is detected or a known one has been modified (e.g., different CPU or memory), Montera is capable of benchmarking it to obtain the previously detailed parameters. Although the employment of benchmarks to profile the *sites* and distribute tasks is not new [30], the incorporation of a tool to dynamically perform this action represents an improvement over previous works.

This benchmarking is fundamental to understand the performance of the infrastructure. Although the first time Montera is executed it presents significant overhead, the information is stored and recovered when needed, so it does not signify a drawback in a long-term approach. Montera also updates the information about the resources after each execution, thus increasing the knowledge about the infrastructure and improving the application performance in real time, all with a negligible computational cost. The bandwidth and whetstones are calculated based on the knowledge about the application -such as the size, model (*constant_effort* and *sample_effort*) and number of samples simulated- and the transmission and execution time of each *chunk*.

The result of the benchmark is also employed to control the availability of the *site*. After three failed benchmarking attempts, the *site* is removed for a fixed period of time from the list of resources to employ. Note that a benchmarking attempt is

considered failed if the expected result -an XML file containing whetstone results and several timestamps- is not provided, thus being able to detect both hardware and software failures. This way, Montera ensures that every task will be submitted to a valid resource, thus maximizing the probability of a successful execution.

The number of slots that were available in past is read at the initiation of Montera to decide the number and size of each task. After it has started and there are *chunks* running in different *sites*, this value is updated in real time. This approach is based on the idea that the number of slots will not vary much with time, so it is a good way of initiating Montera with no overhead and with acceptable precision.

The combination of the proposed characterization and Globus MDS (Monitoring and Discovery System) [47] -employed by GridWay and read by Montera- to gather information about the infrastructure resources is simple yet powerful. Compared with standard monitoring tools such as Nagios [2] or Ganglia [1] it is obviously much simpler and has little capabilities, but fits the purpose it was designed for on a perfect manner. The employment of whetstone benchmark leads to a precise estimation of the resource's capabilities, and storing the resources available on a given moment overcomes the problem of getting trustful information from a user's point of view: for example, knowing that a certain *site* has one thousand CPUs can lead to errors if it has a limit of twenty processes per user, and the existence of hyper-threading on a CPU will result on proving less performance than expected based on its chacrateristics. The proposed approach has also the advantage of not being invasive -not needing to install any software on the remote resource- and distributed, not employing a centralized resource to gather information.

It is noteworthy that Biessel's proposal [7] has been employed to calculate the average and typical deviation of an increasing number of values without the need to storing all of them. Given the average and typical deviation of a set of N elements and a new element n, the authors propose an algorithm to obtain the average and typical deviation of the set composed by the $N + 1$ elements. This way, there is no need to store the entire set of values, and less computational resources are needed to calculate these parameters. It is also remarkable that this approach obtains the mean and typical deviation on a constant time, while traditional formulae scale linearly with the number of elements. Thus, Biessel's proposal represents both performance and storage gains, fully justifying its inclusion in Montera.

3.3 Characterization of the Grid Infrastructure

It has been demonstrated that the performance of a Grid infrastructure can be modeled with only two parameters: *asymptotic performance* and *half performance length* [25, 41]. For the sake of completion, this model will be briefly described here.

From a computational point of view, a Grid infrastructure can be seen as a collection of heterogeneous processors. Therefore, the number of tasks completed in

a given moment can be described as:

$$n(t) = \sum_{i \in G} N_i \lfloor \frac{t}{T_i} \rfloor \tag{2}$$

Where N_i is the number of processors in the Grid infrastructure G that can compute a given task in T_i seconds.

The average behavior of the system can then be represented as the evolution of $n(t)$. If done, a graph similar to Fig. 1 [41] is obtained. Thus, a first-order description of a Grid infrastructure can be made by

$$n(t) = mt + b = r_\infty t - n_{1/2} \tag{3}$$

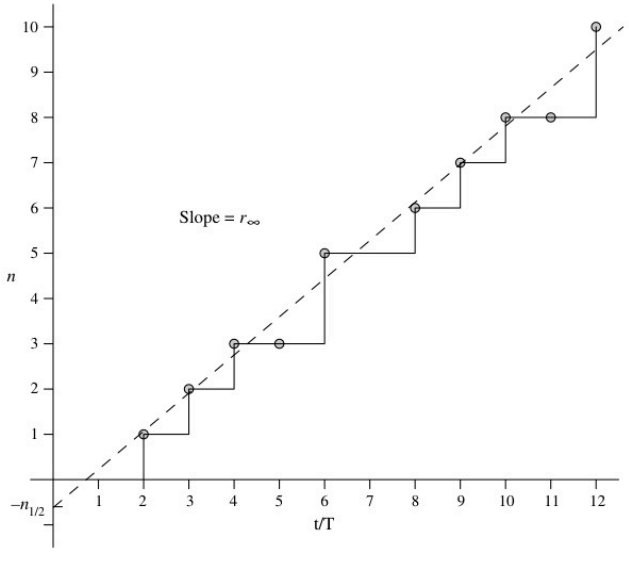

Figure 1. Number of finished tasks as a function of time on an heterogeneous Grid infrastructure

With r_∞ and $n_{1/2}$being the *asymptotic performance* and half performance length as defined in [25]:

- *Asymptotic performance* r_∞: maximum rate of performance in tasks executed per second.
- *Half-performance length* $n_{1/2}$: number of tasks required to obtain the half of the *asymptotic performance*.

The problem with this approach is that it is based on the execution time of constant size tasks, and this work is focused on varying the task size. Thus, modification was necessary to represent the Grid infrastructure behavior when executing

MC codes. This adaptation is proposed here, together with an algorithm to calculate these parameters.

As explained in Section 3.1, MC codes have been modeled according to two parameters, *constant_effort* and *sample_effort*. Depending on the number of samples to simulate, the total computational effort needed to execute the task will vary. Also, the transmission time of the input/output data and queue time must be evaluated, because they represent a different overhead depending on the *chunk* size. All this information must be taken into account when calculating the performance.

Equations 2 and 3 show that it is necessary to know the execution time of each task to calculate the infrastructure performance, and "tasks per second" is employed as measuring unit. The problem with this approach is that Montera is focused on varying the task size, so this cannot be considered static and varies in execution time. Instead, it is now measured in samples/second simulated on the infrastructure. This unit embraces all the aforementioned overheads and accurately models the infrastructure behavior.

This new proposal makes the calculation of the performance significantly more complex. With the Montera approach, it is necessary to know the task distribution in order to calculate the infrastructure performance. However, when employing the DyTSS scheduling algorithm -which will be described in Section 3.4.3- it is necessary to have information about the infrastructure performance to divide the simulation into different *chunks*. To avoid this deadlock, it is necessary to follow Algorithm 1.

Algorithm 1 Description of *create_task_list*() function

```
Require: samples > 0
  siteStatus = read_sites_status()
  taskLlist = create_simple_task_list(siteStatus)
  gridPerformance = calculate_performance(taskList)
  for i = 0 to 100 do
    oldTaskList = taskList
    taskList = DTSS_selfscheduling_algorithm(
      gridPerformance, siteStatus)
    gridPerformance =
      calculate_performance(taskList)
    if taskList = oldTaskList then
      Break
    end if
  end for
  return taskList
```

This algorithm first reads the *site* status (obtained from past executions and the current status obtained with GridWay), calculates a naive task distribution and the infrastructure performance, this is, the ratio of simulated tasks per second.

After that, DyTSS algorithm is employed to calculate a better task distribution. And then again, this task distribution is employed to calculate the *site* performance.

This loop is repeated until an optimum situation without changes between iterations is reached.

3.4 Scheduling Algorithms with Montera

In this work, a new scheduling algorithm called DyTSS (*Dynamic Trapezoid Self-Scheduling*) is proposed. DyTSS is based on the GTSS (*Grid Trapezoid Self-Scheduling*) algorithm and is adapted to focus on the execution of MC codes in extremely dynamic infrastructures.

3.4.1 GTSS Algorithm

The so-called *Grid Trapezoid Self-Scheduler* [22] algorithm is a modification of the *Trapezoid Self-Scheduler* (TSS) algorithm [51], being adapted to a Grid environment. GTSS is based on three main elements: a Grid benchmark model; the relationship factor R_i^{min}; and the TSS dynamic algorithm.

GTSS employs the aforementioned Grid Benchmark model, using r_∞ and $n_{1/2}$ to adjust the *chunk* size. The heterogeneity factor, R_i^{min}, is a coefficient that relates the execution times of every node in a Grid infrastructure. This factor is defined as follows.

$$R_i^{min} = \frac{\overline{T}_{min}}{\overline{T}_{wall}(i)} \tag{4}$$

With $\overline{T}_{min}$ being the minimum of all the execution times.
The behavior of GTSS for a given node j is then defined:

$$C_j^0 = \lceil F * R_{min}^j \rceil \; being \; F = I/4n_{1/2}, \; L = 1 \tag{5}$$

$$C_j^i = C_j^{i-1} - D \; being \; D = \lfloor (F-L)/(N-1) \rfloor \; , \; N = \lceil 2I/(F+L) \rceil \tag{6}$$

Here, F is the size of the first *chunk*, L is the size of the last *chunk*, N is the number of steps, I is the number of tasks to be executed and is the resulting *chunk* size for each *site* j on the i^{th} iteration. F and I can be set statically or can employ the values proposed in Equation 5.

Beyond the mathematical definition, the idea that lies beyond this algorithm is to employ a decreasing number of tasks, balanced depending on the capabilities of each *site*.

The coefficient R_i^{min} represents the relationship between the execution time on the fastest resource and the resource i. Thus, employing it on Equation 5 ensures that the workload of each *site* will be directly related to its performance.

The existence of parameter D to reduce the number of *chunks* along the executions is proven to reduce *makespan*. As has been demonstrated [22], trapezoidal distributions outperform constant ones when dealing with heterogeneous systems.

3.4.2 Problems with pure Self-Scheduling Algorithms in the Grid

During the development of this work, the execution of self-scheduling algorithms on the Grid was intensively studied. In this analysis, a number of significant problems arose.

First, with this type of algorithm, the division of the tasks on different *chunks* is performed at the beginning of the execution. Although this approach is correct in static environments, it is unfeasible in Grid infrastructures, which are, by definition, dynamic [3]. The status of every *site* is regarded as constant along the execution of the simulation, thus obviating an important characteristic of the Grid. Moreover, the relationship between *sites* and *chunks* is not dynamic, so the failure or outage of a single resource results in an unfinished simulation. It can then be concluded that although self-scheduling algorithms perform extremely well in controlled environments, this lack of adaptability makes them sub-optimal in production infrastructures.

To overcome these problems, a new scheduling algorithm, DyTSS, has been created. The technique and parameters employed to divide the simulation into *chunks* is similar to that of GTSS. However, this division is not performed at the beginning of the execution, but every time a new *chunk* is desired. In this way, DyTSS is able to overcome the aforementioned issues, whereas at the same time profiting from the irregular *chunk* distribution of traditional self-scheduling algorithms to make the most of heterogeneous resources.

3.4.3 DyTSS Algorithm

As a particularity of this scheduling approach, the stages of Grid Characterization and Scheduling must be performed simultaneously because of the strong relationship between them. Algorithm 2 shows a high level description of the necessary steps.

First, the information regarding the infrastructure is obtained, as has been already described. With this information, the simulation is divided among all the resources, assigning the same number of samples to each one.

Then, a loop is employed to adjust this initial distribution as follows:

- Calculate infrastructure performance with the current task distribution.
- Apply the GTSS scheduling policy to re-distribute the workload for each resource.

The complexity of this loop is linear on the number of resources. Anyway, given that it is only composed by several basic arithmetic operations per resource, it execution time remains negligible with the size of current Grid infrastructures. Also, a limit of 100 iterations has been established based on the results of several experiments performed during the development of the algorithm: in the case of livelocks, they usually start before the tenth iteration, so the limit was set on the next order of magnitude. Even a finer tune could be performed, the authors consider

Algorithm 2 Description of DyTSS scheduling algorithm

```
Require: desiredSamples > 0
  remainingSampeles = desiredSamples
  taskList = creare_task_list(remainingSamples)
  submit(taskList)
  while remainingSamples > 0 do
    for task in taskList do
      if task finished execution then
        remainingSamples = remainingSamples - task.get_number_of_samples()
        auxTaskList = create_task_list (remainingSamples)
        newTask = auxTaskList.pinck_first_task(task.getResource())
        submit(newTask)
        lastList.add(newTask)
      end if
    end for
  end while
```

that the cost/benefit proportion was favorable to a high limit, ensuring that the loop was not stopped before reaching an equilibrium or livelock.

Note that in this algorithm GTSS is always applied as for C_0^i in Equation 5. The task size decreases because I is reduced any time a previous task has ended, so the steps described in Equation 6 are not necessary.

There is still another innovative approach in the proposed scheduling process. The application of the scheduling algorithm is not employed to obtain the distribution of all the simulations to be performed, but only for a single task for each available slot. Then, when any task ends or a new resource is discovered, the algorithm is applied again, employing real-time information about the *sites* to obtain the size of the next task to be submitted. In this way, the task size is always calculated with the latest information, thus improving the effectiveness of the algorithm. Also, the failure or performance slowdown of any resource does not represent a bottleneck, as happens with the traditional self-scheduling algorithms.

Finally, a technical decision must be noted. The number of tasks submitted to any *site* is not the number of available slots but a 20% greater. This increase is due to two complementary factors.

First, there is a reduction in the execution walltime. If a task is submitted only after the previous one has finished, an overhead is produced due to the result copied back to the local *site*, the current status checked and so. However, if the second task is already queued on the remote *site*, it starts its execution just after the first one ends, thus reducing *makespan*.

It is important to note that task grouping techniques (as described in the previous section) could be used instead of the dynamic creation of jobs in the execution time. The drawback is that, although this approach would reduce the aforementioned overheads, it would represent a flexibility loss. With this approach, the size

of the task is dynamically determined by the status and performance of the entire Grid infrastructure, which could not be done if all the tasks submitted to a single *site* were created at the same time and submitted all together.

The second reason is that the number of available slots in a given *site* can vary during the execution of the application. With this scheduling decision it is ensured that an optimal number of tasks will be created for each resource, while keeping the number of replicas low: if the number of slots is reduced, Montera will detect it and will reduce the number of submitted tasks, cancelling those still waiting so that no CPU time is lost; if it is increased, some of the 20% spare tasks will start their executions, and Montera will detect it and will create more tasks until reaching the 20% excess again. This functionality will be further analyzed in the Results section.

Note that the difference between the proposed scheduling and pure replication algorithms is subtle, but highly relevant. In both approaches, the number of tasks submitted is greater than what is needed, but their behavior is different. In the case of replication, the spare tasks are cancelled, thus losing the computational effort put into their execution. However, in the case of DyTSS -given that in MC codes all results are equally valid- all the results are employed in the simulation, and the only non-useful tasks are those being executed when the desired number of samples is reached. In this way, DyTSS obtains the advantages of replication, such as higher throughput and avoidance of bottlenecks, without suffering the overhead and abuse of the infrastructure inherent to replication algorithms.

3.4.4 Two-level scheduling

As described in Section 2, previous schedulers perform the task scheduling locally and decide the size of the *chunk* to submit to each remote resource prior to the proper execution. In Montera, a new approach is followed, allowing the performance of this scheduling on both the local and remote resources. The objective is to allow the *chunk* to dynamically adjust its size depending on the status of the remote resource and the user needs. To do this, Montera not only submits the desired task, but also includes all the available information about the *site*, the application to be executed, and several scripts in charge of adapting the *chunk* size, depending on the user requirements.

3.5 Montera Architecture

Fig. 2 shows a high-level description of the Montera architecture. As can be seen, the architecture is divided in two different sections: Local Montera and Remote Montera. Local Montera is executed in the local resource, and Remote Montera is executed in the remote resources.

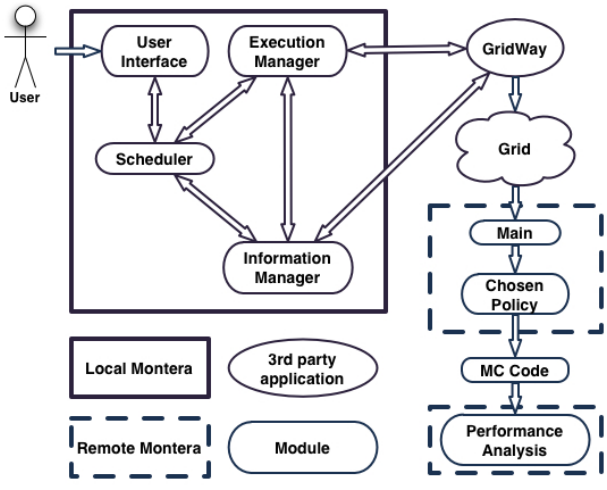

Figure 2. High level description of Montera architecture

3.5.1 Local Montera

Local Montera is the core of the developed framework. Its purpose is to receive the job specifications and requirements from the user and execute them as efficiently as possible.

To obtain the information about the available Grid resources, the Information Manager employs two tools: GridWay [26] and Remote Montera. GridWay is employed to gather the static information the *sites* publish about themselves and the number of tasks being executed at a given moment. Remote Montera provides information about the efficiency of the *sites* after the execution of each task as well as the results of the benchmark executed -if needed- to calculate their real performance.

With this information, the Scheduler decides the ideal size and number of *chunks* to execute on each *site*. Finally, the Execution Manager carries out these executions. It employs DRMAA [50] to perform the job submission and control.

Distributed Resource Management Application API, DRMAA [50] is a high-level API specification for the submission and control of jobs to one or more *sites* within a Grid infrastructure. It defines operations covering all the steps of a Grid execution: submission, monitoring and retrieval of the jobs.

The usage of DRMAA to implement our application allows controlling all the steps involved on the execution of tasks in an unattended way: automatically creating an arbitrary number of tasks, executing them remotely and bringing the results back. To carry this out, only a metascheduling implementing DRMAA API is needed, allowing Montera to delegate these tasks on GridWay.

Montera allows the employment of different scheduling policies. To do this, the Scheduler defines an interface that covers the steps of *chunk* creation and control.

Thus, defining a new policy only requires implementing the interface with the desired functionality.

At this point, it is important to point out that Montera does not strictly depend on GridWay metascheduler but could rely on any other metascheduler implementing DRMAA with slight modifications to the code. Nevertheless, GridWay provides an interesting set of tools that makes it particularly suitable for this work. Also, studies like Poletti's [52] suggest that, in addition to GridWays' capabilities for job management and control, its performance is clearly superior to other widely used tools such as WMS.

An important architectural decision when designing Montera was determining which of the two Grid submission paradigms was more convenient, schedulers or pilot jobs. The problem with pilot jobs is that they currently run only on systems where Network Address Translation (NAT) capabilities are available, preventing the use of software in infrastructures not based on the Grid middleware developed inside the EGEE projects gLite. In addition, since pilot-jobs systems basically act as wrappers for retrieving tasks, they could be also implemented in other way within any scheduler in order to improve the performance efficiency and, at the same time, profit from the migration possibilities, i.e. adapting Montera to any scheduler is possible to profit from the pilot-jobs advantages in a future.

3.5.2 Remote Montera

The second part of the proposed framework, Remote Montera, optimizes the execution of the application on the remote *sites*. Remote Montera is copied to the *site* together with the application to execute, and it decides the exact number of samples that will be simulated in that *chunk*.

Due to the heterogeneity of the remote *sites* and the available tools for each, Remote Montera has been implemented as a set of shell scripts to make it as portable as possible. Its size is greatly reduced, so the overhead induced by its copy to each resource is negligible.

The first script, Main Script, reads the input data, user requirements and information stored regarding the performance of the *site*. Then, the second script, Chosen Policy, calculates the exact number of samples to be simulated and performs the execution. Finally, the Performance Analysis is carried out, and the results are returned to Local Montera.

Montera includes a *Chosen Policy* set to decide the size of each task on each *site*. When the user submits a job to Montera, the user chooses which one to employ, and it will be submitted to the remote *site*. Depending on the desired functionality, the behavior of this script can be completely different. For example, *Deadline* calculates how many samples to execute before a given moment regarding the job submission date, the queue time and the bandwidth. In *Flexible*, Local Montera provides a maximum and minimum number of samples to simulate, and the remote script decides the exact number regarding the *site* performance and its current status. Note that the possibility of combining different local and remote policies allows the

application of widely differing scheduling algorithms, so the user can adapt it to the size and characteristics of the experiment to be simulated.

As in Local Montera, Remote Montera also allows the user's own scheduler to be employed instead of the default schedulers. Creating a new scheduler only requires programming a shell script with the desired behavior. Montera provides information that can be used for this purpose (e.g., submission time, status of the resource) and, if something else is required, it can be provided via the job template.

3.5.3 User Interface

During the development of Montera, much effort was invested in creating an easy-to-use application while at the same time providing the user the necessary tools to implement different policies.

Any scheduling policy must implement an interface with just two methods: `create_chunk_list` and `control_execution`. First, the user defines the number and size of *chunks* to simulate in the first place. Then, the user specifies how the execution will be controlled. The user can check whether the execution of any *chunk* was successful, can create and submit new ones or can cancel the desired ones.

A typical Montera execution is determined by the input file being employed. This file indicates the application to execute, number of samples to simulate, input and output files and the chosen scheduling policy. This is probably the most simple command line-based interface possible, and ensures that any user will be able to perform the desired simulations in a very short amount of time.

3.5.4 Integration of a new application

Given that Montera is designed to execute different applications, the integration of a new one is expected to be seamlessly performed.

Montera expects that the application has a determinate number of parameters, and that they are always located in the same position:

```
<application name> <number of tasks to simulate> <input parameters>
```

If this would not happen the user had to write a wrapper, a task that can be seamlessly performed with a simple shell script.

3.6 Limitations of the Montera approach

As stated in the introduction section, it is important to bear in mind that the proposed code characterization and scheduling algorithm can only be applied to a subset of Monte Carlo codes, the stateless ones. On stateful Monte Carlo codes the problem cannot always be divided on independent executions of arbitrary size, as the result of a given simulation depends on the previous ones. Random Walks, for example, employ random numbers to determine the route of a particle through a user-defined space, and the result of each sample -i.e. one step- depends on its position prior to the advance.

This restriction makes the study of some areas of knowledge beyond the scope of Montera: on protein folding problems, algorithms based on Monte Carlo replica sampling [54], Metropolis Monte Carlo [39] or random walks [42] are employed [44, 13, 58, 6]; lattice QCD [31], employes random walks to determine the position of particles in the space-time; in lattice simulations, Monte Carlo is used for a generation of a probability distribution [35]; in computational chemistry, Metropolis Monte Carlo is widely employed to generate different configurations of the system [29], together with more advanced algorithms [20].

Anyway performing a deep analysis on the usage of different Monte Carlo - based algorithms on scientific applications is beyond the scope of this work. If more information is desired, the proposed references provide a gentle introduction on the area.

4 RESULTS

Here, the results of the Montera scalability and performance analysis will be detailed. To conduct the analysis, a Grid simulator was built to compare the theoretical and practical aspects of the work. Then, several experiments were performed with different scheduling policies and application requirements to demonstrate the feasibility of the proposed approach on a production Grid infrastructure.

4.1 Testbed

The performance analysis of Montera was performed on two Virtual Organizations, namely *fusion* and *prod.vo.eu-eela.eu*. The fusion Virtual Organization, deployed by the EGI Grid Infrastructure (see http://www.egi.eu), counted on 25 *sites* that executed more than 325.000 jobs in 2010 only taking into account European *sites*, which was equivalent to 2,708,319 KSI2K (Kilo SpecInt2000 [43]) hours. Of course, not all of these CPUs can be employed by a single user at the same time because strict usage policies are implemented to prevent abuses. This is a real production environment with diversity of resources (cores from 1 GHz to 3.2, for example), the status os which can be monitored in real time at http://www3.egee.cesga.es. On the other side, GISELA (Grid Initiative for e-Science virtual communities in Europe and Latin America) project deployed an infrastructure oriented to the collaboration between Europe and Latin America research centres, the general purpose VO of which is *prod.vo.eu-eela.eu*. This VO is currently deployed by the Latin American and Caribbean Grid Initiative IGALC (see http://www.igalc.org), which was established also in 2010. Even when there are still not yearly usage statistics, preliminary results [14] show that around 500.000 KSI2K hours were accounted on the first 6 months of existence.

In order to perform the necessary experiments two different MC codes have been employed, namely FANFER2 [49] in its Grid version [45] and ISDEP [10]. They have been chosen because of their deep differences on their behavior. FAFNER2 simulates

many particles on a fast manner, employing several seconds on each one. On the other side, ISDEP needs about one hour on a standard PC to simulate a single trajectory. This way they represent both extremes. If Montera is able to speed both up, it will be proven to be effective on a wide range of applications.

FAFNER2 is a 3D code that simulates the neutral beam injection (NBI) technology on fusion devices. FAFNER2 models the injection of fast neutral particles into toroidal plasmas and the orbits of the resulting ions [34]. Thus, the global efficiency, the losses (i.e., shine-through, charge exchange and orbit losses, including those to limiters), the birth profile and the heating profile can be calculated for different discharges. Because of the design of the code, a single instance of FAFNER2 can only simulate up to 8,000 samples, which establishes an upper bound to the size of each *chunk*.

ISDEP (Integrator of Stochastic Differential Equations for Plasmas) [10] is a MC code that solves the plasma dynamics in fusion devices. It is based on the equivalence between the Fokker-Planck and the Langevin equations. Basically, it simulates the trajectory of fast ions inside the plasma. The MC nature of ISDEP makes it suitable for being executed by means of independent tasks. Therefore, it has been employed in several BOINC initiatives such as Zivis and its followed-up phase Ibercivis, in desktop computing projects such as EDGES and on Grid projects too, for example the EGEE series or EUFORIA.

The local resource on which Montera was executed consisted of a virtual machine running Scientific Linux 4, with GridWay 5.4.0 and Java Virtual Machine 1.5.0.09. This resource was in production status -being employed by different users to perform their daily tasks with a quite restrictive configuration: a scheduling interval of 30 seconds, a dispatch *chunk* of 15 tasks per scheduling interval, and a maximum number of simultaneous jobs per user of 100.

These restrictions do, in fact, limit the scalability of the proposed solution, add overhead and hinder the obtainment of a greater degree of parallelism. The decision to not modify the configuration (i.e., tuning the GridWay configuration to obtain the best possible results) was part of the determination to create an application and scheduling algorithm that outperforms the existing tools in production environments. This particular configuration was established by the system administrators of the local resource as being the most convenient regarding the available hardware and computational demands. Also, this GridWay instance was being employed by several users while the experiment was being conducted. Thus, the possibility of adjusting the system was completely eliminated.

4.2 Simulation of the DyTSS algorithm

Montera includes a simple yet effective Grid simulator. Coupled with the rest of the code, it models the execution of the simulation on a Grid.

This simulator uses the information about the application and the *sites*' performance obtained in past executions of the code, thus being able to accurately calculate the execution time as well as the expected queue time and how long the

data transmission will take. The simulator also employs the number of available resources in the previous executions -which were obtained as explained in the previous section- to calculate the number and size of the *chunks*.

Although the simulator obviates some important characteristics of the nature of the Grid (dynamism, heterogeneity and fault rate), it is still useful for providing an estimation of the performance of any scheduling and as first approach for the initial submission of jobs.

Figures 3 and 4 show the simulation of two different scheduling policies, namely, *EqualChunkSize* and DyTSS. It can be seen that in these simulations the employment of the DyTSS algorithm does not represent a significant improvement over a simple scheduling policy because of the partition of the samples to be simulated in *chunks* of equal size. This is due to the simplicity of the simulation, which was done in a controlled environment: there are no performance losses, new resources or any other issue, so the employed algorithm does not need to adapt the number or size of the *chunks*. As has been explained, one of the advantages of DyTSS over the rest of the scheduling algorithms is its robustness, which is not visible here.

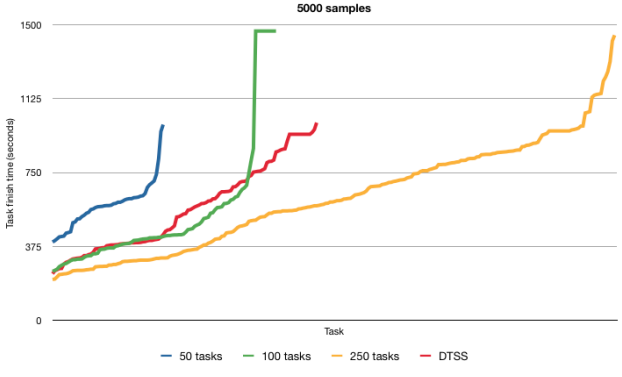

Figure 3. Simulation of a 5,000-sample FAFNER2 execution divided into a different number of equal sized tasks and with the DyTSS algorithm.

As can be seen, with the *EqualChunkSize* policy the results vary depending on the *chunk* size. This is due to the status of the Grid, with about 90 available slots for a single user with the previously described GridWay configuration. In the first case, all 50 tasks start their execution immediately, so there is no queue overhead, and the walltime corresponds to the slowest resource. In the case of the 250-task, there is a significant queue time in two-thirds of the tasks, but having a larger number allows for better scheduling, sending more tasks to the fastest processors. Finally, the 100-task division presents the inconveniences of the two others, but none of their advantages: there are 10 tasks that have a significant waiting time, but they are not enough to perform any kind of useful scheduling. This fact is depicted in the slope of the growing curves. A high or even infinite value indicates that the tasks are waiting for their execution, and low or even null value indicates an immediate

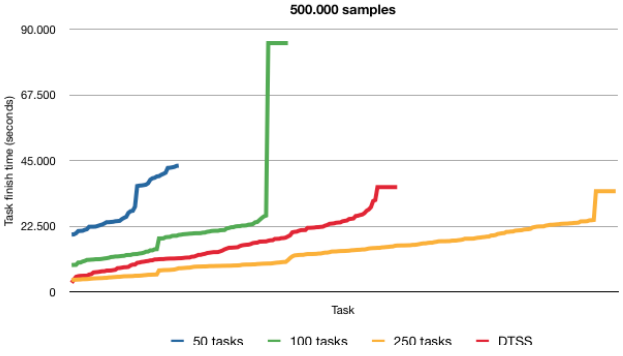

Figure 4. Simulation of a 500,000-sample FAFNER2 execution. Conditions are the same as in Fig. 3

execution of the tasks. DyTSS (Montera) grows with a constant value, so the tasks are progressively executed and, at the same time, the whole calculation lasts for the least possible amount of time.

In other words, in the case of DyTSS, the number and size of tasks has been adapted to the Grid infrastructure, so the execution is always efficient. It is also noteworthy that the DyTSS result graph is nearly a straight line (constant slope) with no vertical steps, as with the others. These steps appear at the point where, for a long time, no tasks have finished, which usually indicates a bottleneck or performance loss.

4.3 Grid Executions

After performing the previous simulations of DyTSS in a controlled environment, different experiments were performed to check the feasibility of Montera in a real Grid environment.

Two different experiments were performed: *Deadline Policy*, where all the different components of Montera were employed to simulate as many samples as possible before a given deadline; and *DyTSS Policy*, where the proposed algorithm is compared with others to demonstrate its superiority.

4.3.1 Deadline Policy

As has been explained, many different scheduling policies can be employed with Montera. In this case, a *Deadline Policy* is proposed, where Montera submits a task to each free slot and executes as many simulations as possible before a given deadline.

This is an interesting policy because it offers a new possibility that, to the authors' knowledge, was impossible to efficiently perform with the previous scheduling

algorithms and tools. This policy is divided into two sections, one in Local Montera and another in Remote Montera.

In Local Montera, a new policy was created. This policy creates one task for each free slot and includes the submission time as an environment variable, which is available in the remote script. In Remote Montera, the remote script that determines the number of samples to simulate reads this environment variable, the local system date, the *site* performance and the application profiling, and it calculates the number of samples that could be simulated before the desired threshold. In this way, the queue time, task cancellation or any other Grid-related issue will have no influence on the accomplishment of the deadline. If a task starts its execution, a certain number of samples will be immediately sampled, and if it stays in the queue for a long time, it will simulate less.

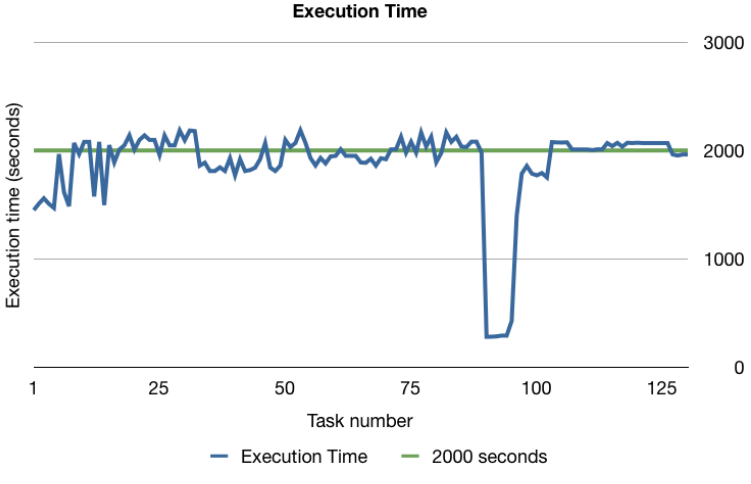

Figure 5. Deadline policy. Task finalization is represented in blue, and the desired ending time (2,000 seconds) is in green.

The results of this policy can be seen in Figure 5, which shows the ending time of every task after requiring Montera to simulate as many samples as possible before a given deadline (2,000 seconds). To provide realistic results, only an execution was performed. Thus, certain tasks failed and most of the others did not perfectly accomplish the deadline requirements. Aside from this small lack of precision - inherent to any Grid execution- it is clear that the proposed characterization of MC codes and Grid *sites* is accurate enough to be used for significant simulations on production infrastructures.

4.3.2 DyTSS Scheduling Policy

Here, the feasibility of the DyTSS scheduling algorithm is proved against two different policies, an *Equal Size* distribution and GTSS. These policies have been chosen

because they are among the most representative: *Equal Size* is a basic policy that is widely employed for these kinds of problems; and GTSS has been demonstrated -as explained in section 2- to be the most efficient self-scheduling algorithm on controlled Grid infrastructures.

In the case of the *Equal Size* policy tests, some adjustments were made to maximize their performance. A rescheduling threshold of 300 seconds and a maximum queue time of 600 seconds were established to avoid saturated resources and to start their execution as soon as possible. Also, the GridWay default scheduling policy -the efficiency of which has been demonstrated [27]- was employed to choose the *site* at which each task is executed. These optimizations were performed to ensure that any improvement obtained with Montera and DyTSS was only due to the application and its scheduling algorithm, not to exogenous factors such as relying on the capability of GridWay to submit and manage the remote execution of tasks.

Some statements about the GTSS tests must also be made. As noted in Section 3.4.2, the information about the infrastructure must be obtained prior to the execution of any self-scheduling algorithm. In this case, the information was obtained with Montera, employing the previously described tools. The alternative option of using public information about the *site* leads to incorrect results and errors. For example, in the local *site* at CIEMAT (ce01-tic.ciemat.es), 208 nodes have been published, but only 20 can be employed by a single user at the same time. If this information alone was employed, the number of submitted tasks would be extremely large, thus affecting the performance negatively.

With FAFNER2 three experiments with increasing task size were performed. The number of samples was 5,000, 25,000, 50,000, 200,000 and 400,000 (from 8 to 180 CPU hours), which can be considered short, medium and long simulations. As can be seen from Fig. 6, different policies lead to different results in terms of *makespan*, and some are more suitable than others, depending on the number of simulations to perform.

In the case of *Equal Size*, results are consistent with those obtained with the simulator (see Fig. 3 and 4 for details). With a small number of particles, the Grid execution tends to benefit from a small number of tasks because the overheads are significant and represent a high proportion of the total execution time. However, when the size of the problem grows, the overhead is not as important. An accurate scheduling is now of vital importance, and it is better performed with a greater number of tasks to distribute.

The results obtained with GTSS were not as good in the real environment as in the simulations. As previously explained, GTSS creates a fixed relationship between tasks and resources. Thus, the failure or performance loss of a single *site* can slow down the entire simulation and increase *makespan*. Although the information obtained with Montera was employed to minimize these issues, the result of employing the algorithm on dynamic infrastructures proved to be suboptimal.

Finally, DyTSS obtained the best results in every single case. From over 650% in the best case to 3% in the worst one, its dynamic task distribution and adaptation to unstable resources resulted in the best alternative among those chosen.

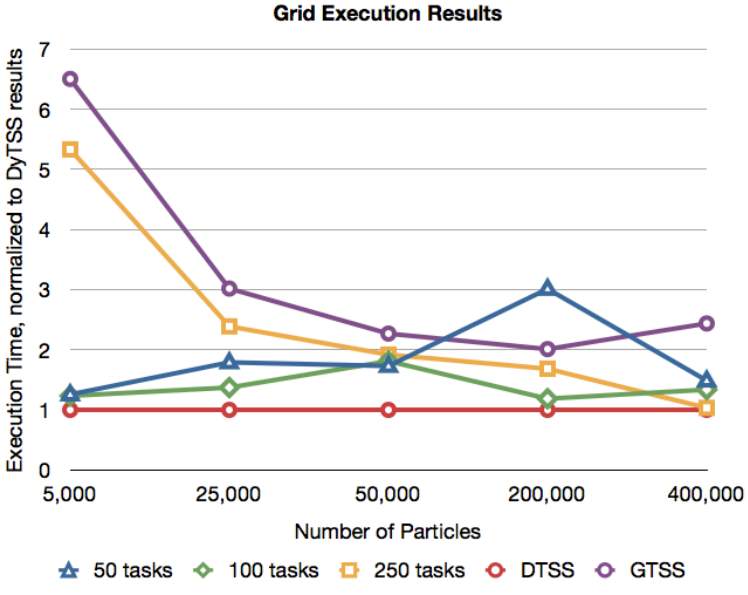

Figure 6. Execution of FAFNER2 with different policies and an increasing number of particles. Results are normalized to DyTSS *makespan*

.

In this experiment, the limit of 400.000 particles is forced by the application design. As it has been stated, FAFNER2 forces an upper limit of 8.000 particles per task, and the smallest distribution with Equal Size has 50 tasks, so the 400.000 particles limit arises. Anyway a typical FAFNER2 experiment employs about 4.000 particles on a serial execution and 8.000 to 80.000 for parallel ones. Thus, demonstrating that DyTSS outperforms all the other algorithms with up to 400.000 particles can be considered significant.

In the case of ISDEP, a minimum and valuable physical research requires of fifty thousand trajectories (50.000). In this paper, results related to 5000 trajectories are presented; such a number has been selected since it clearly represents the benefits for using ISDEP with Montera as a proof of concept without performing the whole research, the results and analysis of which are out of the scope of this paper. The way that such a whole physical research works, which is explained here for the readers' convenience, is that thousands of identical jobs are submitted only differing on a random seed.

The execution of ISDEP presents a particularity that has to be pointed out, also helping to understand the behavior of Montera: depending on some input parameters related to the precision and size of the time steps, the execution time of ISDEP is highly variable, so making no distinction among the different executions would lead on a erroneous profiling and scheduling. Luckily, Montera stores each application

profiling employing a user-defined string name as key. Thus, determining a name like ISDEP_WITH_X_PRECISION to store the application would be enough to ensure that the different ISDEP instances are clustered into groups of similar behavior.

Table 2 shows the results of this execution. Likely to FAFNER2, algorithms GTSS and *Equal Size* with 250 and 500 tasks have been compared with DyTSS.

It is important to notice why in the case of ISDEP the Equal Size task distribution employed have been only those with 250 and 500 tasks. The experiment was designed to be identical as the one with FAFNER2, but a problem arose: most of the Grid *sites* have a limitation on the task execution time, which usually varies between 24 and 48h. Given that each simulation on ISDEP takes about one CPU hour, tasks with more than 20 simulations have high probabilities of being cancelled (and even this phenomenon actually happened in the preliminary tests performed in this work). Thus, the chosen distributions have been 250 and 500 tasks, simulating 20 and 10 particles per task.

For the sake of completion, table 2 also includes the execution time normalized to DyTSS (as in Fig. 1) and the standard deviation.

Samples	*50 tasks (seconds)*	*100 tasks (seconds)*	*250 tasks (seconds)*	*GTSS (seconds)*	*DyTSS (seconds)*
5,000	1331 ± 85,56%	1307 ± 55,77%	5649 ± 65,96%	6892 ± 49,11%	1059 ± 48,91%
25,000	5553 ± 58,79%	5304 ± 62,06%	5568 ± 53,77%	8038 ± 50,03%	2301 ± 43,34%
50,000	5857 ± 61,37%	6131 ± 44,06%	6496 ± 9,88%	7681 ± 15,45%	3385 ± 23,82%
200,000	15665 ± 12,12%	6587 ± 9,07%	10701 ± 29,42%	10456 ± 5,56%	5200 ± 11,57%
400,000	16647 ± 15,17%	14963 ± 25,03%	11568 ± 10,63%	27284 ± 21,98%	11192 ± 12,03%

Table 1. Execution time and standard deviation of the values shown in Fig. 6. Deeper explanation through the text.

Tables 1 and 2 show the 5-averaged execution time and standard deviation for all the experiments performed with FAFNER2 and ISDEP respectively. As can be seen DyTSS always offers the best execution times and usually the most precise results in terms of the execution time. The dynamicity of the Grid infrastructures and variety of resources ensure that the execution time will hardly be the same among different executions, but this standard deviation can give a estimation of how the different algorithms work anyway. A reduced deviation means that the scheduling algorithm is able to overcome small differences on the infrastructure between different executions, and a high deviation means that the result are highly dependent on the specific site executing each task.

It is easy to see why the standard deviation is usually higher when the number of tasks on which the simulation has been splitted is smaller: the number of samples to simulate on each task is bigger and so is the computational effort, so the performance of the *site* executing that task becomes more important. Also, as has been previously described, a smaller number of tasks limits the scheduling possibilities and the obtention of a satisfactory execution time becomes a matter of luck.

parameter	*250 tasks*	*500 tasks*	*GTSS*	*DyTSS*
Execution time [s]	253812	222061	249331	183956
Makespan normalizad to DyTSS	1,35	1,21	1,33	1
Standard Deviation [%]	7,04	32,30	29,68	6,96

Table 2. Results of the simulation of 5000 particles with ISDEP with different scheduling policies. Deeper explanation through the text.

There is still an additional advantage of employing Montera instead a traditional scheduler. As all the decisions about task distribution and scheduling are managed by the application itself, the final user does not need to be aware or spend any time on performing these operations. This represent a significant improvement for final users, who can now focus on their research area and delegate non-essential issues to his specific piece of software.

4.3.3 Dynamic adaptation to free slots with DyTSS scheduling policy

As already noted, with DyTSS Montera is able to detect the number of free slots in any resource and dynamically adapt the number of tasks to be executed. In Fig. 7, a proof of this functionality in a production *site* is graphically detailed.

This experiment consisted on the execution of DyTSS on a single Grid *site*, ce01-tic.ciemat.es. After the benchmarking of the *site*, the execution started with just one task. Montera submitted its 20% more tasks, which was one more in this initial case because Montera adjusted this calculation (number of tasks by 1.2) to the lowest integer. As there are free slots, this second task starts its execution immediately. When Montera detects it, it creates another task to accomplish the 20% margin again, and so on. This process is then repeated until 20 tasks are executed simultaneously, thus reaching the number of free slots that can be employed by a single user on this *site*. Afterwards, the tasks were only created and executed after the ending of the previous ones as no free slots were detected.

When this simulation was finished, the information about the number of free slots was stored. In the future, this information will be employed to determine the number of tasks to be created at the beginning of the execution.

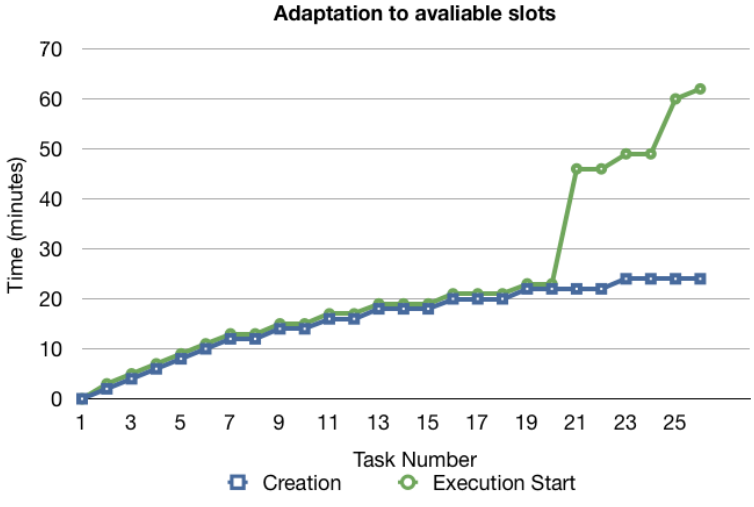

Figure 7. Adaptation to the number of available slots on a *site* with 20 free slots.

5 CONCLUSIONS

In this work, the need for improvement in the efficiency of MC codes on the Grid has been demonstrated.

After presenting the problem, a newly created algorithm called DyTSS is proposed to solve it. This algorithm has the advantage of being focused on a specific kind of application and infrastructure, thus being able to employ a wider set of tools and techniques than general purpose algorithms, such as characterizations of infrastructures, *sites* and applications.

A scheduling application called Montera was then presented. Montera implements mechanisms to collect the aforementioned characterizations, and it provides the user with the ability to perform a two-level scheduling. This approach allows the user to modify the execution depending on his/her needs and the requirements of the experiment.

As has been shown on the Results section, the executions of FAFNER2 with Montera clearly overcome the rest of the alternatives in terms of *makespan*. Thus, it is proven that the employment of Montera is justified for Monte Carlo-based Grid executions.

With MC codes being widely employed in many different areas of knowledge, it is expected that a new tool that simplifies its Grid execution while boosting its performance will be well received in the scientific community. Also, because it is being developed with real world applications in collaboration with users, its correct behavior and usability is guaranteed thanks to the direct feedback between the development team and the final users.

6 ACKNOWLEDGMENTS

The authors would like to thank Mr. Antonio J. Rubio-Montero for the valuable discussions during the development of this work.

References

[1] Ganglia Web: http://ganglia.sourceforge.net

[2] Nagios Web: http://www.nagios.org

[3] I. Foster: What is the Grid? A Three Point Checklist. Grid Today, 1, 22/07/2002 2002.

[4] P. Andreo: Monte Carlo techniques in medical radiation physics. Physics in medicine and biology, 36:861, 1991.

[5] M. Arellano and S. Bond: Some tests of specification for panel data: Monte Carlo evidence and an application to employment equations. The Review of Economic Studies, 58(2):277–297, 1991.

[6] U. Bastolla, H. Frauenkron, E. Gerstner, P. Grassberger and W. Nadler: Testing a new Monte Carlo algorithm for protein folding Proteins, 32(1), 52–66, 1998.

[7] H. Biesel: Recursive calculation of the standard deviation with increased accuracy. Chromatographia, 1977.

[8] P. P. Boyle: Options: A Monte Carlo approach. Journal of Financial Economics, 4(3):323–338,1977.

[9] S. Brooks: Markov chain Monte Carlo method and its application. Journal of the Royal Statistical Society: Series D (The Statistician), 47(1):69–100, 1998.

[10] F. Castejón et al: Ion kinetic transport in the presence of collisions and electric field in TJ-II ECRH plasmas. Plasma Physics Controlled Fusion, 49:753–776, 2007.

[11] A. T. Chronopoulos, M. Benche, D. Grosu, and R. Andonie: A class of loop self-scheduling for heterogeneous clusters. Proceedings of the 3rd IEEE International Conference on Cluster Computing, pp. 282–291, 2001.

[12] H. J. Curnow and B. A. Wichmann: A syntetic Benchmark. Computer Journal, 19(1):43–49, 1976.

[13] C. Czibula, M. Bocicor and I-G Czibula,: Solving the Protein Folding Problem Using a Distributed Q-Learning Approach. International JOurnal of Computers, 5(3):404–413, 2011.

[14] R. Diacovo: Gisela Infrastructure Status Report, EU Deliverable: D4.1. Universidade Federal do Rio de Janeiro, pp. 1–14, 2010.

[15] P. Diaconis:The markov chain monte carlo revolution. American Mathematical Society, 46:179–205, 2009.

[16] J. Díaz, S. Reyes, A. Niño, and C. Muñoz-Caro: Derivation of self-scheduling algorithms for heterogeneous distributed computer systems: Application to internet-based grids of computers. Future Generation Computer Systems, 25(6):617–626, 2009.

[17] F. Dong and S. G. Akl: Scheduling algorithms for grid computing: State of the art and open problems. School of Computing, 2006.

[18] R. Eckhard: Stan Ulam, John Von Neumann and the Monte Carlo Method. Los Alamos Science, Special Issue, 1987.

[19] Y.-W. Fann, C.-T. Yang, S.-S. Tseng, and C.-J. Tsay: An intelligent parallel loop scheduling for parallelizing compilers. Journal of Information Science and Engineering, (16):169–200, 2000.

[20] D. Ferguson and. J. Siepmann: Monte Carlo methods in chemical physics (p. 467). Kluwer Academic Publishers, 1999.

[21] J. Hernández et al: CMS Monte Carlo production in the WLCG computing grid. Journal of Physics: Conference Series, 119:052019–052028, 2008.

[22] J. Herrea Sanz: Modelo de programación para infraestructuras Grid computacionales. PhD thesis, Universidad Complutense de Madrid, Madrid, 2009.

[23] J. Herrera, E. Huedo, R. Montero, and I. M. Llorente: Loosely-coupled loop scheduling in computational grids. Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, p. 6, 2006.

[24] D. B. Hertz: Risk Analysis in Capital Investment. Boston, MA : Harvard Business Review, 1964.

[25] R. W. Hockney and C. R. Jesshope: Parallel Computers Two: Architecture, Programming and Algorithms. 1988.

[26] E. Huedo, R. Montero, and I. M. Llorente: The GridWay framework for adaptive scheduling and execution on grids. Scalable Computing: Practice and Experience, 6(8), 2005.

[27] E. Huedo, R. Montero, and I. M. Llorente: Evaluating the reliability of computational grids from the end user's point of view. Journal of Systems Architecture, 52(12):727–736, 2006.

[28] S.-H. Jang, X. Wu, and V. Taylor: Using performance prediction to allocate grid resources. Technical report, 2004.

[29] W.L. Jorgensen and J. Tirado Rives: Molecular modeling of organic and biomolecular systems using BOSS and MCPRO. Journal of Computational Chemistry, 26(16):1689–1700, 2005.

[30] G. Kitagawa: Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. Journal of computational and graphical statistics, 5(1):1–25, 1996.

[31] G.P. Lepage: Lattice QCD for Novices. ArXiv High Energy Physics - Lattice e-prints, 2005.

[32] C. Li and L. Li: Utility-based QoS optimisation strategy for multi-criteria scheduling on the grid. Journal of Parallel and Distributed Computing, 67(2):142–153, 2007.

[33] Y. Li and M. Mascagni: Grid-based Monte Carlo Application. Grid Computing—GRID 2002, pp. 13–24, 2002.

[34] G. Lister: FAFNER: A Fully 3-D Neutral Beam Injection Code Using Monte Carlo Methods. 1985.

[35] J. Lukkarinen: Lattice Simulations of the Quantum Microcanonical Ensemble. ArXiv High Energy Physics - Theory e-prints, 1997.

[36] L. Maigne, D. Hill, P. Calvat, V. Breton, R. Reuillon, Y. Legre, and D. Donnarieix: Parallelization of Monte Carlo simulations and submission to a Grid environment. Parallel processing letters, 14(2):177–196, 2004.

[37] L. Malcolm, L. Spaulding Eric, Anderson, I. Tatsu, and H. Eoin: Simulation of the oil trajectory and fate in the arabian gulf from the Mina Al Ahmadi spil. Marine Environmental Research, 36(2):79–115, 1993.

[38] M. Mascagni and Y. Li: Computational Infrastructure for Parallel, Distributed, and Grid-based Monte Carlo Computations. Lecture Notes in Computer Sciences, 2907:39–52, 2003.

[39] N. Metropolis and A. Rosenbluth: Equation of state calculations by fast computing machines. The journal of Chemical Physics 6 (21):1087–1092, 1953.

[40] N. Metropolis: The Beginning of the Monte Carlo Method. Los Alamos Science,Special Issue, 1987.

[41] R. Montero, E. Huedo, and I. M. Llorente: Benchmarking of high throughput computing applications on grids. Parallel Computing, 32(4):267–279, 2006.

[42] K. Pearson: The problem of the random walk. Nature, 72:294, 1905.

[43] H. Renshall: New Time Units simplify procedures. CERN computer newsletter, 2004.

[44] Rocha, L. F. O., Tarragó Pinto, M. E., and Caliri, A. :The water factor in the protein-folding problem. Brazilian Journal of Physics, 34, 90101, 2004.

[45] M. Rodríguez-Pascual, J. Guasp, F. Castejon Magana, A. J. Rubio-Montero, I. M. Llorente, and R. Mayo Garcia: Improvements on the Fusion Code FAFNER2. IEEE Transactions on Plasma Science, 38:2102–2110, 2010.

[46] R. Roveda, D. B. Goldstein, and P. L. Varghese: Hybrid Euler/Direct Simulation Monte Carlo Calculation of Unsteady Slit Flow. Journal of Spacecraft ad Rockets, 37(6), 2000.

[47] J. M. Schopf, M. DArcy, N. Miller, L. Pearlman, I. Foster, and C. Kesselman: Monitoring and Discovery in GT4: Functionality and Performance. Technical report, 2005.

[48] D. Silva, W. Cirne, and F. Brasileiro. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. Euro-Par 2003 Parallel Processing, pp. 169–180, 2004.

[49] A. Teubel, J. Guasp, and M. Liniers: Monte Carlo simulations of NBI into the TJ-II Helical Axis Stellarator. Technical Report 4, 1994.

[50] P. Tröger and P. Domagalski: Standardization of an API for Distributed Resource Management Systems. Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007), pp. 619–626, 2007.

[51] T. H. Tzen and L. M. Ni: Trapezoid Self-Scheduling: A Practical Scheduling Scheme for Parallel Compilers. IEEE Transactions on Parallel and Distributed Systems, 4:87–98, 1993.

[52] J. L. Vázquez-Poletti, E. Huedo, R. Montero, and I. M. Llorente: A comparison between two grid scheduling philosophies: EGEE WMS and Grid Way. Multiagent and Grid Systems, 3(4):429–439, 2007.

[53] J. L. Vazquez-Poletti, E. Huedo, R. Montero, and I. M. Llorente: CD-HIT Workflow Execution on Grids Using Replication Heuristics. Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on, pp. 735–740, 2008.

[54] R. H. Swendsen and J.-S. Wang:Replica Monte Carlo simulation of spin glasses. Physical Review Letters, 57(2), 2607–2609, 1986.

[55] M. Wu and X. Sun: Grid harvest service: a performance system of grid computing. Journal of Parallel and Distributed Computing. 66(10):1322–1337, 2006.

[56] C.-T. Yang and S.-C. Chang: A parallel loop self-scheduling on extremely heterogeneous PC clusters. Proceedings of International Conference on Computer Science, pp. 1079–1088, 2003.

[57] C. Yu and D. C. Marinescu: Algorithms for Divisible Load Scheduling of Data-intensive Applications. Journal of Grid Computing, 8(1):133–155, mar 2010.

[58] , Y. Zhang, D. Kihara and J. Skolnick: Local energy landscape flattening: Parallel hyperbolic Monte Carlo sampling of protein folding. Proteins, 48(2):192–201, 2002.

Manuel Rodríguez-Pascual has a Master Degree in computing sciences from the Universidad Complutense de Madrid. Presently he is a PhD candidate at CIEMAT, where he is involved in several international R&D projects. His interests are in the areas of application optimization for grid computing, scheduling and middleware development.

Dr. Ignacio M. Llorente has a graduate degree in physics (B.S. in physics and M.S. in computer science), a Ph.D. in physics (Program in computer Science) and an Executive Master in business administration. He has about 15 years of research experience in the field of High-Performance Parallel and Distributed Computing, Grid Computing and Virtualization. Currently, he is a Full Professor in Computer Architecture and Technology at Universidad Complutense de Madrid, where he leads the Distributed Systems Architecture Group.

Dr. Rafael Mayo García is Ph.D. in physics by the Universidad Complutense de Madrid-UCM, Madrid (Spain) since 2004 and has obtained several postdoctoral fellowships such as Marie Curie and Juan de la Cierva Grants. He has been working as Researcher at UCM in experimental and Computational Plasma Physics. Since 2005 he works at CIEMAT (Madrid) in the ICT Division on Supercomputing and Grid developments. Dr. Mayo belongs to the Spanish Royal Society of Physics and the Latin American Bioinformatics Society. He is author of more than 60 publications and conference presentations and has been involved in 20 international and national R&D projects where he has also held managerial and coordinating activities.

Improvements on the Fusion code FAFNER2

Manuel Rodríguez-Pascual, José Guasp, Francisco Castejón, Antonio Juan Rubio-Montero,
Ignacio Martín Llorente and Rafael Mayo

***Abstract*—FAFNER2, a 3D code adapted to the TJ-II helical axis stellarator from the original one developed by Lister at Max Planck IPP, simulates by Monte Carlo methods the Neutral Beam Injection (NBI) technology (a key heating method for most of the fusion experiments worldwide).**

To the date, FAFNER2 has been usually run at CIEMAT by means of a batch mode on a shared memory computer with MIPS processors. This work describes how the application has ben updated to employ MPI and run on standard Linux clusters. As in Grid infrastructures not every site has MPI installed on their nodes, a serial version has also ben developed, together with a Java DRMAA program able to run FAFNER2 on distributed resource platforms, such as Grid infrastructures. In addition, and with new improvements in the code for maximizing its performance, the metascheduler GridWay has been also incorporated for maximizing the executions on the Grid. Scalability, fault tolerance and correctness of the result have been proved employing a significant number of particles and nodes within a local cluster and the EGEE infrastructure.

***Index Terms*—FAFNER2; NBI heating; Grid applications.**

I. Introduction

NUCLEAR fusion could be an energy source by employing a mixture of deuterium and tritium plasma under extreme temperature and pressure. Several techniques for heating this plasma are used in the fusion experiments. Thus, the plasma can be heated with waves in the Ion Cyclotron Range of Frequencies (ICRF), by means of the Electron Cyclotron Resonance Heating (ECRH) affecting (only) the electrons, or by the Neutral Beam Injection (NBI) heating.

The NBI method is a widely employed technique to heat the plasma in fusion experimental devices. For the sake of evidence, it has been chosen as one of the heating methods for ITER [1]. The method is rather simple: neutral atoms are not affected by the confining magnetic field of a tokamak or a stellarator and are ionized in the plasma via collisions with ions and electrons. The generated fast ions are also confined in the magnetic field and are able to exchange their energy to plasma ions and electrons. Typical injection energies are in the range of 40 KeV to 130 KeV (for comparison, the central plasma thermal energy can range form about 1 KeV to 15 KeV). The generation of fast neutral atoms occurs in three steps: generation of a powerful ion beam in the range of several MW; neutralization in a gas target; and, transport of the neutral atoms to the torus, deflection of the non-neutralized ions to a so-called ion dump.

Nuclear fusion still presents some issues to become a commercial energy source. The main of them are technological, but there are some aspects of Plasma Physics that must be addressed. Computer simulations could help to overcome this problem.

There exist software applications that simulate different sections and phenomena of a fusion reactor whose main motivation is to help in fundamental understanding and to guide ideas on designing new devices that could become profitable. With them, it is then possible to run simulations as experiments at less cost and faster than building new facilities. Even more, lately it will be also possible to check in real time the expected evolution of the plasma or try to predict which will be the further interesting conditions to run experiments. An example of this is the computations that are being carried out with the DKES code [2], the results of which can be useful to efficiently design magnetic coil systems on stellarators

In this context, FAFNER [3], an application that simulates NBI heating, can be a powerful tool for fusion physicists. It can be employed by its own or coupled to other codes, thus creating complex workflows that can simulate a wider range of phenomena inside the plasma. Even more, the computational simulation for the optimization of an NBI heating mechanism is not only of outmost importance for the most possible efficient design, but for the three-dimensional Monte Carlo shielding analyses on the ITER NBI duct for the nuclear and Bremsstrahlung radiation [4]. It is also necessary to minimize radiation effects induced by the degradation of solid insulating materials under a fusion radiation environment or by insulating gas which will be required around the NBI high voltage feed line, ion source and accelerator [5].

The only drawback is that the kind of software that can perform such calculations needs a lot of computing resources to produce valid results in a reasonable time period. In this context, Grid infrastructures represent a powerful tool for the fusion community, as their members are able to share their equipment and obtain a big pool of computational resources. Several international projects like EGEE (see http://public.eu-egee.org.) or EUFORIA (http://www.euforiaproject.eu) already provide the physical resources -both CPU time and storage- needed for the efficient execution of fusion applications. EGEE, for example, provides 45 nodes with more than 15000 CPUs through its fusion Virtual Organization, the computation of which arose to more than 2500 KSI2K (Kilo SpecInt2000 [6]) hours during 2009 only in European sites.

Grid computing has already shown its capability to be used to solve several types of problems that appear in fusion plasma physics and that involve distributed calculations. Monte Carlo

Manuel Rodríguez-Pascual, José Guasp, Francisco Castejón, Antonio Juan Rubio Montero and Rafael Mayo García are with CIEMAT, Avda Complutense 22. 28040 MADRID (Spain). e-mail: (manuel.rodriguez (at) ciemat.es, guasp (at) ciemat.es, francisco.castejon (at) ciemat.es, antonio.rubio (at) ciemat.es, rafael.mayo (at) ciemat.es)

José Guasp and Francisco Castejón Magaña are with Laboratorio Nacional de Fusión, Avda Complutense 22. 28040 MADRID (Spain)

Ignacio Martín Llorente is with DSA-Research.org, Universidad Complutense, C/ Prof. José García Santesmases s/n. 28040 MADRID (Spain). (e-mail: llorente (at) dacya.ucm.es)

codes -the one that FAFNER is based on- and parameter scan problems are among the most suitable for this purpose, as they require high computation time and the problem can be divided into fully independent tasks.

The aim of this paper is to explain how the adapted version of FAFNER to the helical stellarator TJ-II [7] has been revised in order to improve its local execution and how it has been modified to be also executed on Grid and to be coupled to new codes such as ISDEP [8], so an analysis on the new physics that can be studied from now on is also present.

To achieve this, two complementary versions of the application have been developed. The first one consists on an updated version of FAFNER2 [7], so it can run on modern clusters and High Performance Computers (HPC): SHMEM [9] has been replaced by MPI [10] as a message passing mechanism; MIPS by X86 as the processor architecture; and, Irix64 by Linux as Operating System. Then -as MPI could present some drawbacks on Grid Computing- a serial version of FAFNER2 has been created. Finally, a DRMAA [11] application has been developed to run on the grid multiple concurrent executions of this serial version, creating in this way a new FAFNER2 code which will be able to be coupled to ISDEP. GridWay [12] metascheduler has been employed to monitor and control these executions.

II. The FAFNER code

In nuclear fusion devices, NBI heating is one of the important ways of heating the plasma to the temperatures required for the onset of nuclear fusion processes. In some instances, neutral injection is used with hydrogen in order to avoid neutron emission and induced radioactivity. This generates a rather high fraction of hydrogen in the plasma in discharges with NBI, so diverse studies have been carried out to analyze the isotope effects on plasma confinement [13] in order to control the H/D isotope ratio on nearly pure deuterium plasmas.

The physical problem to solve then is the modeling of neutral beam injection into three-dimensional toroidal plasmas, and to calculate the trajectory of the resultant fast ions until they get lost or are absorbed by the system.

The FAFNER code simulates the injection of fast neutral particles into a toroidal plasma and the orbits of the resulting ions [3]. Thus, the global efficiency, the losses (i.e. shine-through, charge exchange and orbit losses, including those to limiters), the birth profile and the heating profile can be calculated for different discharges. For doing so, the input required data are the field configuration, the measured density and temperature profiles and the beam parameters. Instead of real space coordinates, in a second version of the FAFNER code the guiding centre part in magnetic coordinates is treated, so toroidal magnetic flux Ψ and toroidal and poloidal angles Φ and Θ respectively [14] are used. The use of magnetic coordinates of the Boozer type has the advantage of providing a simple representation of the magnetic geometry together with the fact that a natural representation of quantities, which are constant on magnetic surfaces (e.g. the electric potential, densities and temperatures), is provided. The guiding centre approximation allows the separation of the slow motion across the field lines from the fast motion along the field lines. Additionally, FAFNER can also gives the initial pitch (the ratio between the parallel and the total velocity) of each particle and the energy conservation is inherently fulfilled.

The FAFNER2 code used in this work [7] is a Monte Carlo which is an updated version of the original one and includes charge-exchange processes, limiter effects and a new method of selecting an appropriate subset of guiding centre particles. Results for calculated orbits with this version as well as birth and heating profiles or the influence of radial electric fields on the heating efficiency of neutral beam injection [15] were found to be in good agreement with experiments once they were simulated with this updated code.

A special case of interest is the application of FAFNER2 in the TJ-II device,a flexible heliac axis stellarator whose configuration is produced by three different sets of coils: the central conductor, composed of a circular and a helical coil, the toroidal and the vertical field coils.This is so because of the high toroidicity, helical excursion, high values of rotational transform and great multiplicity of local magnetic wells, which make very difficult to perform reliable analytical or semi-analytical calculations. For this reason, numerical simulations performed with the updated version of FAFNER have been carried out [16], [17].

From a computational point of view, the code is constituted by two different sections. The first one simulates the injection of neutral particles into the torus. As a result, the coordinates and speed of these particles are obtained. After that, in the second section, this information is used to determinate the initial ionization of these atoms inside the plasma, and then, the energy distribution of the plasma as a result of their interaction with the fast ions. FAFNER2 also computes the energy lost as a result of the collisions between the neutral beam and the scrappers and ducts.

From a mathematical point of view, Monte Carlo techniques are employed to compute the coordinates of a set of fast ions, corresponding to a neutral beam, as well as the appropriate plasma parameters. The trajectory of these ions and its interaction with the plasma are described in terms of a Fokker-Planck equation. The trajectories of the fast ions are solved with a Boozer flux coordinate system. These numerical techniques are well known and have been previously employed to describe heating of neutral beams in stellarators, but nowadays in tokamaks.

FAFNER2 provides two sets of data on each execution. First, the position and speed of a given number of particles is determined. Then, it obtains different statistics about plasma heating process such as the fraction of ionized particles, absolute power provided to the torus, and fraction of lost particles.

III. The Computing Grid

When trying to execute an application with massive computational needs, Grid computing arises as one of the most cost/performance effective solutions, specially when dealing with loosely coupled applications. Nevertheless, it is important

to note that an agreement on the definition of "Grid" is far from being achieved among the scientific community.

An intuitive definition of the Grid could be: while the World Wide Web is a service to share information on the Internet, the Grid is a service to share computational and storage resources on the Internet.

In a classical article by Ian Foster [18] a three-point checklist is proposed to determine whether a given system can be considered a Grid infrastructure or not. Regarding this list, it is a system that:

- Coordinates resources that are not subject to centralized control.
- Uses standard, open, general-purpose protocols and interfaces.
- Deliver nontrivial qualities of service.

The first point means that the resources can belong to different organizations, each one with different software, usage and security policies. This resource sharing must be highly controlled, so users, service providers and infrastructure administrators are aware of *when*, *to whom*, *for what* and *under which conditions* a certain resurce is shared.

By requiring that the protocols and interfaces (*middleware*) are general-purpose the resulting infrastructure is not oriented to solve a certain kind of problem and/or application, but can be employed to different areas of knowledge. The employment of standard, open protocols encourages the users to adapt the tools to their specific needs, while avoiding fragmentation.

And of course, the resulting infrastructure must be reliable and stable enough to constitute a valid alternative to the users, so they can carry on their experiments.

Grid computing has already showed its potential in several scientific fields [19], [20] such as High Energy Physics, Bioinformatics, Chemistry, Earth Sciences and Fusion. For the sake of evidence, a project like EGEE has performed almost $5 \cdot 10^8$ KSI2K hours during 2009 only taken into account the four Large Hadron Collider experiments.

IV. Implementation

In this section the adaptation of FAFNER2 in order to make it suitable to an efficient Grid execution is described.

The module associated to the TJ-II geometry is independent from the code, what enables the code to be used for different fusion experimental devices.

A. MPI Application

Since the creation of FANFER, back in 1987, the original code has been periodically updated in order to make the most of the existing hardware platforms. The last version running at CIEMAT relied on SHMEM message passing toolkit, as it was designed to run on a SGI Origin 3800 shared memory server. Nowadays this involves a serious drawback, because HPC an Grid computing is usually performed on clusters with X86 processors.

1) Message passing mechanism update: from SHMEM to MPI: The first task was to change the message passing mechanism, from SHMEM to MPI. [21] SHMEM is a proprietary technology present in some SGI systems, but not on standard clusters. As the final goal is to run FAFNER2 on the Grid, it is mandatory to use standard technologies, and MPI is the most widely used within EGEE sites.

SHMEM library is based on MPI version 1, and in many cases their routines are equivalent. Thus, the first step was to find and replace these routines. Then, the application was optimized in order to avoid unnecessary inter-task communications and barriers, and to take advantage of all the possibilities that MPI provides by employing the most suitable function to each situation. As a consequence, execution time was slightly reduced. The results are shown in section V-B1.

As Input/Output is not exactly the same on SHMEM and MPI, it was also necessary to adapt the code in order to have exactly the same output files. These files are employed as entry files in other applications, and any difference will make them fail.

2) Architecture update: from MIPS to X86: After porting the code to MPI, the application was migrated from its obsolete MIPS architecture to the X86 platform.

The first step consisted of deleting obsolete or unnecessary packages, replacing them with newer versions. The size of the executable was also minimized by removing all the unused routines from the source code. In the case of NAG mathematical library, its size has been reduced by a 99%.

Also, a small number of routines utilized in FAFNER2 do not belong to Fortran standard, but to a private library implemented for a certain architecture and operating system, present on the computers when the original version of the program was created, back in 1994. When migrating the application these routines became unavailable or had a slightly different functionality. Thus, it was necessary to locate and implement them again, in order to make the application work exactly as expected.

Originally, FAFNER2 was designed to be executed on 64 bits processors. Nevertheless, as EGEE hardware and operating system usually are 32 bits, it is necessary to make the application runnable with this word size. The original data size was maintained both for integer and real variables in order to obtain the same precision, although this situation caused unalignment of some variables. To solve this issue their declarations were re-ordered, and then alignment was forced by employing an specific flag on the compiler.

When this point was reached, the application was ready to be run on standard computers. Its performance is analyzed in section V-B2;

B. Serial DRMAA Application

Distributed Resource Management Application API, DRMAA [11], is a high-level API specification for the submission and control of jobs to one or more sites within a distributed memory infrastructure. It abstracts the programer from the job management tasks, by defining operations covering all the steps of a Grid execution: submission, monitoring and control of the jobs.

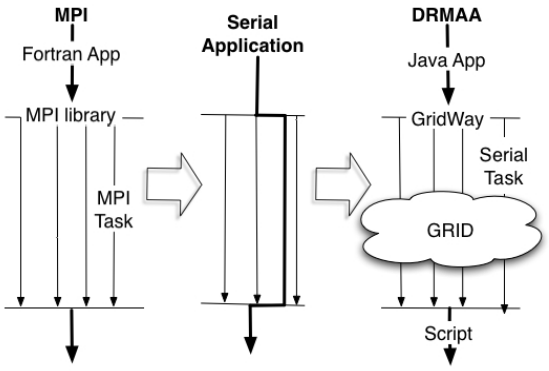

Fig. 1. Architecture of MPI, Serial and DRMAA versions of FAFNER2

By employing DRMAA to implement our control application it is possible to automatically create any number of tasks, execute them remotely and obtain the results. Depending on the chosen scheduler, the same application can be executed on private resources (SGE, Condor) or the Grid (GridWay).

The substitution of MPI by DRMAA in FAFNER2 was accomplished in two steps.First, MPI was removed from the code, thus obtaining a serial application. Then, a Java DRMAA application was implemented. It employs GridWay to submit serial the FAFNER2 version to the Grid and recover the results.

On the MPI version of FAFNER2, when the application starts, the first task checks the input data, initializes certain data structures and perform calculus common to all tasks. Then, the work is split among all the tasks. Being a Monte Carlo code there is no need of interprocess communication, so each one is fully independent. After all the tasks have finished working, the first one recovers the partial results and combine them to obtain the output of the application.

In this work, this behavior is emulated with the following tools: a serial application, a DRMAA application, shell scripts, and the GridWay metascheduler. It is important to point that, with this process, the same results as in the MPI version are achieved, thanks to the post-process script that combines the partial results produced in the different Grid sites.

1) From MPI to Serial code: The first step consisted on removing the message passing mechanism. The goal here was to create a serial application that behaves as one specific task of the MPI version, determined with an input parameter.

Of course, the sections of the code that were only computed by the first task part must now be performed by every instance: input data must be read, data structures initialized, results of the execution written to a file, and so on. Consequently the final result is an application that behaves like the first MPI task on some parts of the code, and like any other task during the rest of the execution. Fig. 1 represents this process.

In order to combine the results of several FAFNER2 executions, the code was slightly modified. FAFNER2 produces output data in two different places: along the program execution, and after performing all the calculations. This information has been processed when it is shown on the screen or stored in a file (for example, statistical analysis have been performed), the application had to be modified in order to provide raw data, before being processed.

To achieve this target, some sentences that export the required variables and partial results to an XLM file were introduced into the code. This way, besides having a fully functional application, it produces results that are employed on the gridification. These files, together with the program output, constitute the input of the post-process script. This script reads all the XML files, and together with the information of the output files generates a new output file.

2) DRMAA application: The next step was to create a DRMAA application which employs the Grid to execute multiple, concurrent instances of the serial version of FAFNER2.

This DRMAA application receives two parameters as input: the input data of FAFNER2, and the number of instances to be executed. With this information it creates a template and submits the jobs to the Grid. It waits until all jobs have finished, retrieves the results from each one, and combines them to obtain the final result employing the aforementioned script.

3) Metascheduler: By employing DRMAA, it is ensured that the code can be coupled to different schedulers. In this case, GridWay has been the chosen one. It is a metascheduler that highly simplifies Grid execution of jobs by automatically performing the steps involved in job submission: system selection, system preparation, submission, monitoring, migration and termination [22]. It provides some key features required by FAFNER2:

- Straightforward DRMAA connection. After linking the DRMAA library of GridWay with our Java DRMAA application, the submission and control of the grid executions is delegated to GridWay. It choses the best node, submits the application, monitors the execution and retrieves the results.
- Scheduling capabilities: GridWay employs a dynamic scheduling system. It can detect when a new machine has been added or removed from a the considered Grid infrastructure, and redistribute the work load. Also, it detects the chages in the services and computational resources provided by any site.
- Fault detection & recovery capabilities. Transparently to the end user, GridWay is able to detect and recover fromthe failure, outage or saturation of any Grid element.

V. Experimental Results

When modifying an application, specially after the performed change of paradigm, it is mandatory to compare the performance of both old and new versions. In this section the different versions of FAFNER2 were executed with a fixed set of input data in order to obtain reliable statistics about its execution.

A. Testbed Description

The behavior of the DRMAA FAFNER2 implementation has been analyzed on a testbed based on several local clusters and on the EGEE Grid infrastructure (through the *fusion* VO).

The main characteristics of the local clusters are described in Table I.

TABLE I
CHARACTERISTICS OF THE TESTBED RESOURCES

Name	Characteristics
Jen50,	122 cores MIPS R14000@500 MHz, 126 GB total
Lince, (184 slots)	46 nodes Dual Xeon *Nocona* 3.2 GHz, 2 GB each 1Gb/s Ethernet
ce01-tic.ciemat.es	60 CPUs Dual-core 5160 Xeon 3 GHz 2 GB each 1Gb/s Ethernet

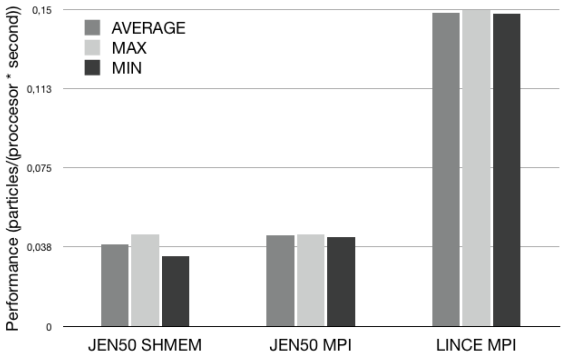

Fig. 2. Performance of SHMEM for MIPS, MPI for MIPS and MPI for X86 versions of FAFNER2

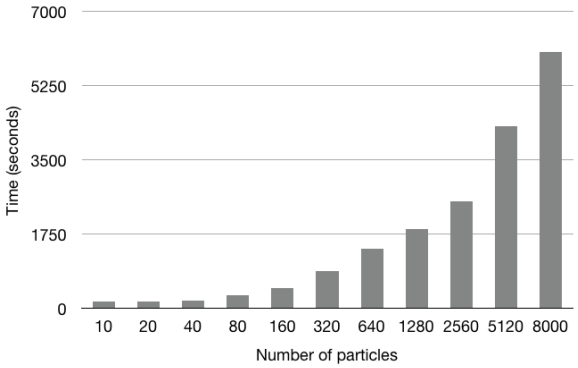

Fig. 3. Scalability of FAFNER2 in terms of the number of particles

B. MPI version performance

Here, the improvements in terms of performance and execution time after the changes in the application depicted in section IV-A are detailed.

1) SHMEM to MPI: First, the performance gain obtained after upgrading the message passing mechanism was analyzed. Both versions of FAFNER2 were executed on the same computer, a Cray T3E called *Jen50*, under different load conditions to achieve realistic results. Fig. 2 shows the average performance of these two versions measured in particles per processor second, i.e., the number of simulations that a single processor can carry out in a second.

As can be seen, after migrating the application from SHMEM to MPI and minimize its network traffic the average performance has been increased in about 10%. Also, the difference between the fastest and slowest execution has been reduced.

2) SGI to X86: The next step was to upgrade the code to the new architecture X86 from the previous SGI one. It implies several changes in the code due to the difference between several FORTRAN and MPI libraries or the recompilation of the NAG library among others. Then, the performance in both platforms was compared. The obtained results, also shown on Fig. 2, are impressive: it can be appreciated that the performance of the application has been multiplied by a factor of three. This improvement shows how necessary it was to update FAFNER2 in order to run on modern X86 clusters.

C. DRMAA version performance

With the change of the computational paradigm, from MPI on cluster to DRMAA on Grid, a complete analysis of the new application must be carried out. In the following sections, the performance, scalability and overhead of the proposed solution have been measured.

1) Scalability: First, the scalability of the new application was studied. The application is expected to work on Grids, hence it is important to test its performance under heavy work demands. These tests were performed in two folds: number of particles/task, and number of slots. To study the scalability in terms of number of particles a local cluster was employed, in order to have a controlled, well known environment. Here, one single instance of the serial version of FAFNER2 was executed with different number of particles. Results (see Fig. 3) show that the execution time and the number of particles grow in the same proportion.

Here, two things must be taken in mind. First, the time of checking input data and generating output results is approximately constant -about 150 seconds- and that is why the results of 10 to 40 particles look identical; second, that the precision is one second, hence there results from the lower part of the graphic are not entirely accurate.

Then, an analysis on the scalability of FAFNER2 in terms of number of Grid slots was performed. Results are detailed on Fig. 4, and show that the execution time is roughly constant after 10 executions/each. Notice that the first cases (1 to 8 slots) perform better than the rest. That is due to GridWay's scheduling capabilities: as it prioritizes the best resources to execute a job, the first ones usually get better sites than the rest of the jobs.

2) Resource Availability: When evaluating the differences between MPI and DRMAA versions of FAFNER2, it is important to check in which resources will they be able to be executed.

As it has been discussed in the previous section, scalability of DRMAA version is only limited by the available resources, confirmed by the experimental results shown in Fig. 4. In this case, a resource is any *fusion* VO site, because a serial job with no library dependences can be executed virtually everywhere.

Fig. 4. Scalability of FAFNER2 in terms of the number of Grid slots

On the other side, MPI version of FAFNER2 is restricted to sites with MPI installed, and in addition its number of threads is limited by the number of slots of the remote site.

Table II shows the state of the *fusion* VO while these tests were carried out (July 2009). The first column shows both the sites with and without MPI support, and the reminder only sites with MPI.

As explained, serial jobs of the DRMAA application do not need MPI support, hence can be executed on any of the 3587 slots belonging to the 33 available sites. On the other side, MPI jobs need a site with, at least, as many slots as the the number of desired tasks. Table II shows the limitation of its scalability in this particular infrastructure: running a 100-task MPI job needs an MPI-enabled site with at least 100 slots, what is only possible in five sites, and a 500-task MPI job can only be executed in three sites -two of them belonging to the same research center-. Moreover, it will probably be quicker to allocate 500 DRMAA (serial) jobs than a 500-task MPI job, as it is hard to find a site with such amount of free resources.

3) Overhead: When evaluating the performance of the application, the overhead induced by the changes in the code and additional software and hardware layers must be measured.

Code serialization has an obvious problem. The operations and checkings that were executed only once in the MPI version, now must be executed in all tasks. This represents an overhead of approximately 60 seconds per task in the cluster *Lince*.

The execution of FAFNER2 on remote sites also demands the copy of the input data and application executable, as well as the recovery of the partial results.

These two overheads scale linearly with the number of nodes. They are both parallelizable: it is possible to transmit several files simultaneously without performance loss (depending on the network connection of both local and remote *sites*) and, once transferred, executions on the different *sites* are concurrent. Thus, their influence on the total execution time is very small.

The overheads produced in the local site must also be taken into account. The first one is produced by the DRMAA application. Its execution time is about one second, so we consider it does not affect scalability. Also, GridWay metascheduler has a scheduling interval of 30 seconds -by default- and a dispatch chunk of 15. This could represent a scalability issue if the simulation was divided in a high number of very small tasks, but otherwise is negligible.

The overhead produced by post-process script is more important. When executing FAFNER2 with a high degree of parallelization, this is a factor that must be taken into account, since it will increase linearly with the number of tasks. The final average is of two seconds/task.

4) Speedup: A detailed analysis on the different steps that have been taking in order to create a new FAFNER2 code has been done. However, a comment on the time gains obtained from using the Grid is still needed.

As the reader can see from the previous results present in this section, this improvement raise up to near 400%. This estimation refers to a comparison between the old and the new MPI versions of FAFNER2 in terms of the code performance, i.e. the number of particles calculated per processor per second. If this comparison is done between the MPI (local cluster) and DRMAA (Grid) versions, It can be found that the first calculates the same number of particles (8000, the heaviest load used in this work) only 1.7 times faster than the second with a serial single task. With an intermediate particles number such as 2560, the ratio is 1.3. This demonstrates that the Grid version is suitable to be used since it has no restriction in the number of tasks to be submitted as it has been mentioned in the scalability subsections, i.e. there are no restrictions as there usually are in the supercomputers use policy. More evident is the comparison between MPI and DRMAA versions to be run on the Grid, since the first is very difficult to allocate in Grid sites with enough number of MPI free slots (see Table II).

D. Analysis on the physical results

At this point of the work it has been proved that FAFNER2 is suitable to run on a Grid infrastructure, but an analysis on the correction and accuracy of the physical results obtained is still remaining. To overcome this issue, a two-fold simulation has been performed: first, an increasing number of tasks with the same number of particles simulated on each one were submitted to the Grid; then, the total number of particles was maintained constant while varying the size of each task. These statistics are useful for the final user since they allow him/her defining the calculation to be performed. For many NBI uses cases, a certain number of particles could produce the same accurate and reliable value than a calculation with a higher one in a lower time.

Fig. 5 shows the obtained physical results of one FAFNER2 output value: the proportion of total/lost particles. The number of particles simulated per task remains constant, so a higher number of tasks implies a higher number of particles simulated. As the number of tasks increases, the final proportion of total/lost particles reach a threshold that can be taken into account by the user in his/her future executions. For the sake of comparison and estimation of the required accuracy by the user too, an additional value related to the Students-t deviation is also plotted.

A second parameter is also analyzed in Fig. 6. As can be seen, the value related to the proportion of total/lost particles by FAFNER2 is used again, but with respect to the number of particles calculated per task. In this case another two values have been included, namely percentage of total absorption and total injected power.

TABLE II
SCALABILITY LIMITATION IN *fusion* VO DEPENDING ON THE DEGREE OF PARALLELISM

Resource	Serial	50 task MPI	100 task MPI	500 task MPI	1000 task MPI
Sites	33	6	5	3	0
Slots	3857	2413	2355	2100	0

Note 50 task MPI column means that there were 6 sites where 50 tasks for an MPI job could be executed and so on.

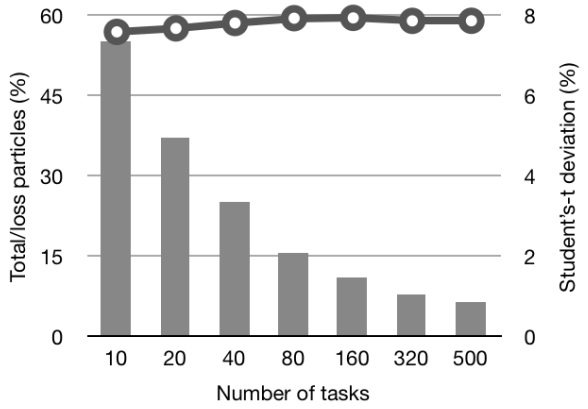

Fig. 5. Evolution of the FAFNER2 parameter "Proportion of total/lost particles" with the number of tasks submitted (dots). The accuracy of the results by means of the Students-t value is also depicted (bars)

In this test the total number of particles remains constant to 8000, so simulating 10 particles per task implies 800 tasks and so on. With this test, a demonstration of how the application can be executed on the Grid with an increasing value of parallelism is proved. In this case, it is noticeable that the typical deviation is reduced when the size of the chunk is bigger, and Students-t deviation increases in a similar proportion. This is due to the nature of the application: FAFNER2 returns the average value of the aforementioned parameter, and then both deviations are calculated. Increasing the number of particles on each task means that the average value returned will be more precise, so the typical deviation is smaller, but having an lower number of tasks increases Students-t deviation. Thus, for a final evaluation, both parameters should be taken into account.

For the sake of completion, the last part of this development consisted on performing an analysis of the influence of random numbers on FAFNER2. To do so, the FORTRAN default generator was replaced by a C one. The resulting differences were negligible, confirming the feasibility of this approach.

VI. FUTURE WORK

FAFNER2 is an application that will evolve on the following years. Besides being employed at CIEMAT to simulate the processes inside TJ-II, both on Grid and on MPI clusters, it is expected to simulate NBI heating at ITER. To perform this task, FAFNER2 code must be adapted, providing a clear and easy-to-use interface.

When this tasks is accomplished, a web interface to manage FAFNER2 and analyze its results will be created, in order to simplify the usage of the application. This interface will also allow coupling FAFNER2 to other fusion codes such as ISDEP [8], creating complex *workflows* that simulate a wide number of plasma phenomena.

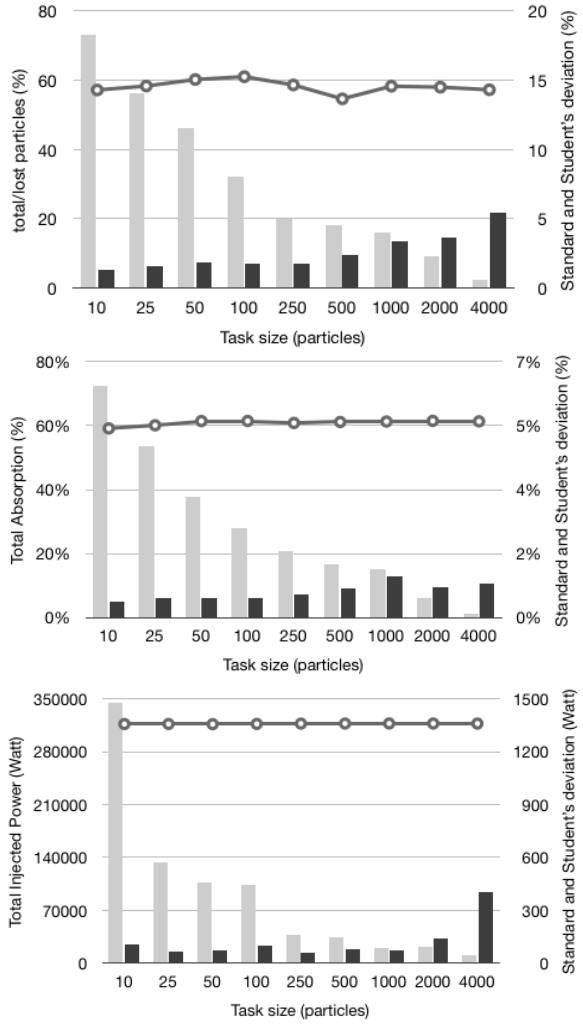

Fig. 6. Evolution of three FAFNER2 parameters with the size of the tasks submitted (dots). The accuracy of the results by means of the Students-t value (dark grey) and standard deviation (light grey) are also depicted.

VII. NEW PHYSICS INSIGHT COMING FROM THE PORTING OF FAFNER2

Having FAFNER2 running on the grid allows the user to have quick results that can be used as input for other applications, thus allowing the possibility of establishing complex workflows among several fusion codes, which open the door to explore a new range of physical problems. Two examples are detailed here.

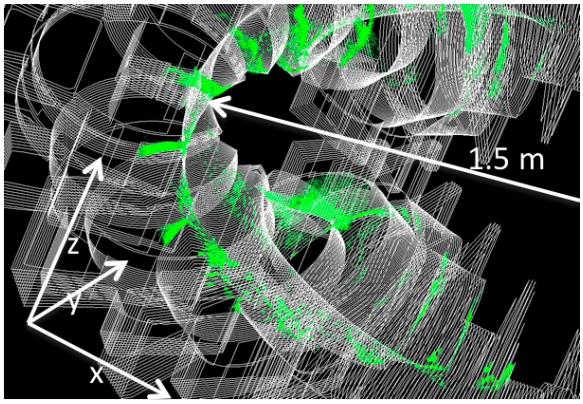

Fig. 7. Striking points of fast ions with the vacuum Wessel. It is also possible to see the size of the chamber.

FAFNER2 provides the results of the interaction of the launched neutral atoms with the background plasma. This implies the solution of the linearized kinetic equation that can be approximated by a Fokker-Planck equation. As FAFNER2 works, the background plasma characteristics are constant, hence the linear character of the problem. In particular, an important quantity provided by the code is the absorbed power that has a strong influence on the plasma evolution. The latter is estimated by a transport code that solves a set of equations, including the heat transport one:

$$\frac{3}{2}n(r,t)\frac{dT(r,t)}{dt} - n\chi \bigtriangledown T(r,t) = P_{FAFNER}(r,t) - P_{LOSSES}(r,t) \quad (1)$$

Where T is the temperature, n is the density, P_{LOSSES} is the lost power due to radiation and recombination processes, and P_{FAFNER} is the power delivered to the plasma by the energetic neutral atoms, estimated by FAFNER2. All these quantities are functions of the time and the position. Up to know, this calculation has been performed in a non-satisfactory way, having a database of results for different plasma parameters [23]. The problem is that the final results strongly depend on the precise shape of the plasma profiles, not only on a single parameter, therefore, the database should be huge in order to cover all the possible plasma profiles. As P_{FAFNER} depends on the plasma characteristics, must be evaluated from time to time and the most appropriated way to do this is using the Grid version.

Another utility of the Grid version of FAFNER2 is to use the code output as an input for ISDEP [8], which is a Monte Carlo global code that follows ion trajectories in the plasma background. This problem was solved by using FAFNER2 to estimate the birth positions of the fast ions in the plasma, as input for ISDEP [24]. This calculation produces a three dimensional map of the striking points of fast ions on the vacuum vessel of the device (See Fig. 7). The figure shows a 3D Plot of the vacuum chamber with the striking points of the ions. The size of the chamber is also shown: 1.5 m from the centre of the torus to the groove of the Wessel. Indeed, the Grid version will allow just to plug ISDEP to FAFNER2 and to start following neutrals that become ions when they collide with the plasma background. These ions will then be followed by ISDEP. So the Grid version of FAFNER2 allows plugging these two codes in a natural way and to obtain the properties of the fast ions dynamics in the background plasma.

VIII. Conclusions

During this work, the following fundamental objectives have been: upgrading FAFNER2 to state of the art technologies, create a serial version of the application, and create a DRMAA application to control Grid execution.

The first task, updating the application from SHMEM to MPI, was successfully achieved. Although it was necessary to make deep changes in the code, the execution time was cut down to about one third, maintaining data precision and the same output format. As the proposed solution is based on standard, open-source solutions, the resulting application is expected to have a longer lifecycle and simpler upgrading process.

Then, a serial version of FAFNER was created by removing MPI from the code. Together with the aforementioned DRMAA application, a high degree of parallelization (up to 1024 tasks) has been reached, demonstrating the feasibility of the proposed approach. A deep analysis of the induced overheads has been carried out.

By providing two complementary versions of FAFNER2, the final user can now decide which one to execute, depending on his/her specific needs and available computing resources: MPI on his local cluster, and/or Grid DRMAA when the resources from the Grid are needed. The decision about it usually refers to the available resources in the first case, the number of which is limited by the cluster administrator; thus, there are several restrictions such as a fixed number of instances to be run by the user or a wall time limit per job. Similar restrictions are also present in the Grid, but they have a lower impact in the final user since the total pool of resources is much bigger. This point is of importance since it will allow final users better addressing their needs and better using the computational infrastructures, so any effort in porting a code to new computational platforms becomes rapidly worthwhile. In addition, the new simulations that can be now done will help in the building of new, more efficient, fusion devices.

Last but not least, the new physics that can be studied from now on by coupling the new FAFNER2 code to other such as ISDEP would have been impossible to do without the solutions provided in this work.

References

[1] N. Miyamoto *et al*, "Experimental results on ITER-NBI concept source," *AIP Conference Proceedings*, vol. 380, pp. 300–308, 1996.

[2] A. J. Rubio-Montero *et al*, "Drift Kinetic Equation Solver for Grid (DKEsG)," *IEEE Trans. Plasma Sci.*, 2010. doi: 10.1109/TPS.2010.2055164

[3] G. Lister, "FAFNER: A fully 3-d neutral beam injection code using monte carlo methods," Garching: Max-Plank Institut für Plasmaphysik. Tech. Rep. 4-222, 1985.

[4] S. Sato, H.Iida, M. Yamauchi, and T.Nishitani, "Shielding design of the iter nbi duct for nuclear and bremsstrahlung radiation," *Radiation Protection Dosimetry*, vol. 116, no. 1,4, pp. 28–31, 2005.

[5] E. R. Hodgson and A. Moroño, "Radiation effects on insulating gases for the ITER NBI system," *Journal of Nuclear Materials*, vol. 258-263, pp. 1827–1830, 1998.

[6] H. Renshall, "New time units simplify procedures," *CERN Computer Newsletter*, vol. 39, no. 2, p. 4, 2004.
[7] A.Teubel, J.Guasp, and M. Liniers, "Monte carlo simulations of NBI into the TJ-II helical axis stellarator," Garching: Max-Plank Institut für Plasmaphysik, Tech. Rep. 4/268, 1994.
[8] F. Castejón, L. A. Fernández, J. Guasp, V. Martin-Mayor, A. Tarancón, and J. L. Velasco, "Ion kinetic transport in the presence of collisions and electric field in TJ-II ECRH plasmas," *Plasma Phys. Control. Fusion*, vol. 49, no. 6, pp. 753–776, 2007.
[9] K. Feind, "Shared memory access (SHMEM) routines," *CUG 1995 Spring Proceedings*, pp. 203-208, 1995. .
[10] A. Geist *et al*, "MPI-2: Extending the message-passing interface," *Euro-Par'96 Parallel Processing*, pp. 128-135, 1996.
[11] P. Troger, H. Rajic, A. Haas, and P. Domagalski, "Standardization of an API for distributed resource management systems," *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*, pp. 619–626, May 2007.
[12] E. Huedo, R. S. Montero, and I. M. Llorente, "The GridWay framework for adaptative sheduling and execution on Grids," *Scalable Computing – Practice and Experience*, vol. 6, no. 3, pp. 1–8, 2006.
[13] M. Bessenrodt-Weberpals *et al*, "The isotope effect in ASDEX," *Nucl. Fusion*, vol. 33, p. 1205, 1993.
[14] A. Boozer, "Guiding center drift equations," *Phys. Fluids*, vol. 23, pp. 904–909, 1980.
[15] A. Teubel and F. Penningsfeld, "Influence of radial electric fields on the heating efficiency of neutral beam injection in the W7-AS stellarator," *Plasma Phys. Control. Fusion*, vol. 36, p. 143-152, 1994.
[16] J. Guasp, M. Liniers, C. Fuentes, and G. Barrera, "Thermal load calculations at TJ-II vacuum vessel under neutral beam injection," *Fusion Technology*, vol. 35, pp. 32–41, 1999.
[17] J. Guasp and M. Liniers, "Loss cone structure for ions in the TJ-II helical axis stellarator. part I: properties without a radial electric field," *Nuclear Fusion*, vol. 40, p. 397-409, 2000.
[18] I. Foster, "What is the grid? a three point checklist," *Grid Today*, 2002.
[19] F. Berman, G. C. Fox, and A. J. H. (eds), *Grid Computing: Making The Global Infrastructure a Reality*. Chichester: John Wiley & Sons Ltd, 2005.
[20] P. Herrero, M. Pérez, and V. Robles (eds), *Scientific Applications of Grid Computing*. Springer Berlin Heidelberg, 2004.
[21] Message Passing Interface Forum, "Document for a standard message-passing interface." Knoxville: University of Tenneessee, Tech. Rep. CS-93-214, 1994.
[22] E. Huedo, R. S. Montero, and I. M. Llorente, "Grid resource selection for opportunistic job migration," *Lecture Notes in Computer Science*, vol. 2790/2004, pp. 366–373, 2003.
[23] V. I. Vargas *et al*, "Density dependence of particle transport in ECH plasmas of the TJ-II stellarator", Madrid: Informes Técnicos CIEMAT, no. 1162, 2009.
[24] F. Castejón *et al*, "Flux-expansion divertor studies in TJ-II," *Nuclear Fusion*, no. 49, pp. 085019–085026, 2009.

Manuel Rodríguez-Pascual has a Master Degree in computing sciences from the Universidad Complutense de Madrid.

Presently he is a PhD candidate at CIEMAT, where he is involved in several international R&D projects. His interests are in the areas of application optimization for grid computing, scheduling and middleware development.

Dr. José Guasp has a PhD in physics from the Universidad Complutense de Madrid.

Presently, he has a permanent position in the National Fusion Laboratory at CIEMAT where has been involved in many computational and experimental fusion plasma experiments for more than 40 years. He has published 45 papers in referenced scientific journals in Plasma Physics and controlled Fusion.

Dr. Francisco Castejón Magaña has a PhD in physics from the Universidad Complutense de Madrid.

Presently, he has a permanent position in the National Fusion Laboratory at CIEMAT and is the Head of Plasma Theory Unit and leader of the fusion area in EGEEIII. He has worked at Caradache (France) during 1988 and at Khpti in Kharkov (Ucraine) in 1992. He has participated in three expert *adhoc* groups and nowadays is member of the Steering Committee of the European Fusion Development Agreement. He has published more than 90 papers in international scientific journals and more than 200 presentations to International Congresses in Plasma Physics and controlled Fusion. He has acted as supervisor of 5 PhDs.

Antonio Juan Rubio Montero received the M.Sc. in computer science from the Universidad Complutense de Madrid (UCM), Spain, in 2003.

He has occupied different positions as programmer and systems architect in several private companies before he was hired as Researcher on Grid Technologies at CIEMAT, Madrid, in 2006. He is author of more than 20 publications focused on Grid application porting and Cloud computing, which were resulted from his participation in several international and national Grid initiatives and from his relation with the GridWay Team (UCM) as external collaborator. Also, he is in charge of maintaining the Grid sites and participates in the administration of the computational resources devoted to scientific research belonging to the ICT Division at CIEMAT.

Dr. Ignacio M. Llorente has a graduate degree in physics (B.S. in physics and M.S. in computer science), a Ph.D. in physics (Program in computer ccience) and an Executive Master in business administration.

He has about 15 years of research experience in the field of High-Performance Parallel and Distributed Computing, Grid Computing and Virtualization. Currently, he is a Full Professor in Computer Architecture and Technology at Universidad Complutense de Madrid, where he leads the Distributed Systems Architecture Group.

Dr. Rafael Mayo García is Ph.D. in physics by the Universidad Complutense de Madrid-UCM, Madrid (Spain) since 2004 and has obtained several post-doctoral fellowships such as Marie Curie and Juan de la Cierva Grants.

He has been working as Researcher at UCM in experimental and Computational Plasma Physics. Since 2005 he works at CIEMAT (Madrid) in the ICT Division on Supercomputing and Grid developments.

Dr. Mayo belongs to the Spanish Royal Society of Physics and the Latin American Bioinformatics Society. He is author of more than 60 publications and conference presentations and has been involved in 20 international and national R&D projects where he has also held managerial and coordinating activities.

More efficient executions of Monte Carlo Fusion codes by means of Montera: the ISDEP use case

M. Rodríguez-Pascual, A.J. Rubio-Montero, R. Mayo
CIEMAT
Madrid, Spain
Email: {manuel.rodriguez, antonio.rubio, rafael.mayo}@ciemat.es

A. Bustos, F. Castejón
Laboratorio Nacional de Fusión
Asociación EURATOM/CIEMAT
Madrid, Spain
Email: {andres.debustos, francisco.castejon}@ciemat.es

I. M. Llorente
DSA-research.org
Madrid, Spain
Email: llorente@dacya.ucm.es

***Abstract*—ISDEP (Integrator of Stochastic Differential Equations for Plasmas) is a Monte Carlo code that solves the plasma dynamics in a fusion device and perfectly scales on distributed computing platforms. Montera is a recent framework developed for achieving Grid efficient executions of Monte Carlo applications, as ISDEP is. In this work, the improvement of performing the calculations of ISDEP with Montera, which rise up to 34.9%, is shown as well as an analysis on the implications it could have, which aim to show to the fusion research community the benefits of using Montera.**

***Keywords*-Montera; WMS; Grid; Plasma Dynamics;**

I. Introduction

One of the main issues in the fusion research field is that the plasma (ionized gas) inside a fusion device is modeled with rather complicated equations, most of them without analytical solutions. The accurate knowledge of ion trajectories in low collisionality regimes for both tokamaks and stellarators in order to understand the confinement in those devices is a must. The trajectories of particles have then been estimated in stellarators as a first assessment of the confinement properties of a given magnetic configuration [1], but also in tokamaks [2]. Computationally, this problem can be overcome by an ion kinetic calculation based on Monte Carlo (MC) techniques, which solves Langevin equations in plasmas.

Because of this, Castejón *et al*. developed the Integrator of Stochastic Differential Equations in Plasmas (ISDEP) code [3], which was used to estimate the ion collisional transport for the TJ-II device without the assumption of radially narrow particle trajectories [4]. Later on, and also in TJ-II, the experimentally observed increase in the ion temperature in the electron cyclotron heating regime during the transition to the core electron-root confinement (CERC) regime was confirmed to be attributed to the joint action of the electron-ion energy transfer (which changes slightly during the CERC formation) and an enhancement of the ion confinement [5]. This improvement, related to the increase in the positive electric field in the core region, was confirmed by means of calculations performed with ISDEP code by estimating the ion collisional transport in TJ-II under the physical conditions established before and after the transition to CERC.

All of these calculations were performed on distributed platforms. The MC nature of ISDEP makes it suitable for being executed by means of independent tasks. Therefore, it has been employed in several BOINC initiatives such as Zivis and its followed-up phase Ibercivis, in desktop computing projects such as EDGES and on Grid projects too, for example the EGEE series or EUFORIA.

Related to the Grid environment, the first ISDEP version, ready for stellarators, was firstly ported inside EGEE. After that, a new version adapted to tokamaks was ported in the framework of EUFORIA. For doing the latest, a non-linear heating term had to be included, i.e. the quasi-linear waveparticle interaction that causes plasma heating was added, so heating and collisional transport could be analyzed on the same level and modifications of transport by heating could be estimated too. It can be inferred then that the development of ISDEP is still not closed and it is open for several improvements as it will be explained later. Consequently, a huge use of the code is expected and, here, Grid computing can provide part of the computing time required.

Nevertheless, the Grid executions are far from being optimum beyond the data challenges because of the dynamic characteristics that this environment has. To date, several techniques have been developed in the scheduling of jobs on the Grid, but all of them have significant drawbacks. They are either of general purpose, omitting important issues related to the state, characteristics and continuous changes in the resources and how they influence on the makespan of the application to execute; or, they have not been applied to a standard Grid infrastructure since their sphere of action was too concrete. Even more, most of

the tests performed with these scheduling techniques have been done on simulators which, sometimes, do not count on realistic production environments with dynamic work loads, changing resources and incomplete information. Thus, the real capacities that these kinds of computational platforms offer are not currently being employed at their best.

In this context is where Montera (MONTE carlo RApido, Fast MC from its Spanish translation) [6] aims to be useful to the Grid community. Specifically devoted to MC codes, it takes into account the continuous changes that are produced in a dynamic production Grid infrastructure in order to better execute the different tasks that are submitted by adapting its number and distribution to the available resources. To do so, realistic characterizations of every site in every moment as well as a profiling of the used code that is being employed are the pillars for its operation.

II. The ISDEP code

Under several approximations, the plasma can be described with a Fokker-Planck Equation (FPE). This FPE is a 5D partial derivative equation for the plasma distribution function, impossible to be solved mathematically and very difficult to be solved numerically. ISDEP [3] is based on the equivalence between the FPE and the Langevin equations. Langevin equations are stochastic differential equations for a single particle motion, instead of the whole plasma.

The development of the computer code ISDEP was done in order to solve the guiding centre equations in the presence of collisions with ions. The guiding-centre (GC) approximation [7] is considered for allowing one to disentangle the fast gyromotion from the relatively slow displacement of the particle along the device. Instead of the position and velocity of the physical particle, one considers the position and velocity of the guiding centre of its fast Larmor precession. Ion dynamics are represented by a closed set of five coupled stochastic differential equations (SDE), namely:

$$\frac{d\vec{r}_{GC}}{d} = \vec{a}_{r_{GC}} \tag{1}$$

$$\frac{d\lambda}{dt} = [a_\lambda + a_\lambda^{Ito}] + b_\lambda \xi_\lambda \tag{2}$$

$$\frac{dx^2}{dt} = [a_{x^2} + a_{x^2}^{Ito}] + b_{x^2} \xi_{x^2} \tag{3}$$

where both ξ magnitudes represent independent white noises. As discussed in [3], this system represents an Itô diffusion process of test ions in a background plasma.

At the beginning of the process, the field particles are described by experimental density, temperature and electrostatic potential profiles and electrons. Lately, in order to estimate the heat transfer from electrons to ions, the collision operator was extended following [8], for taking into account the collisions of the test ions also with electrons (see appendix A in [5] for details).

More details about the computational process and the considered physical assumptions (out of the scope of this work) can be found in [3] and [5], including details of the integration of long trajectories in the complex TJ-II magnetic configuration. The statistical analysis of many particle trajectories is used in ISDEP to obtain physical quantities of the whole plasma, like energy, confinement time, escape points of the ions, etc.

As aforementioned, the continuous development of the ISDEP code brings the possibility of facing new and more ambitious problems. In addition, its deployment on several computational platforms allows widening of its use and impact. Grid technology could be a main actor in these actions. New features such as the adaptation of the code to ITER, keeping the non-linear and the heating terms or the introduction of the Alfvén resonances in the code, can be successfully done by means of Grid calculus. But also, ISDEP is a useful tool to establish complex workflows between grid codes, but also between Grid and HPC applications.

III. The Montera framework

Grid is not a totally controlled environment actually. Many sites, allocated at different places all around the world, putting into practice common use policies, but not always standards, relying on a great diversity of computing and storage resources, executing a wide set of codes and applications, etc. lead the final user to a conclusion: a more efficient execution based on the code that is using in a concrete moment when it is submitted should be an asset. Also, the code, whatever it would be, has its own characteristics, which are of importance in order to profit as much as possible from the employed computing platform.

Up to the moment, a huge effort has been put on the improvement of Grid performance, with many different approaches focusing on different concepts of Grid infrastructure, the particular characteristics of the application to be executed or the available and published information. Anyway, most of these developments do not take into account how the infrastructure could evolve in time, do not bear in mind the software being executed or have been tested only in controlled environments or simulators.

MC codes are extensively employed in Science, and their easy distribution in independent tasks makes them ideal for Grid Computing. That is why Montera, see [6] for a deeper explanation, aims to fill the lack for a specific tool to execute MC based applications on the Grid. Montera also relies on three complementary tools, employed to gather information from different sources, which is lately used to distribute the samples to be simulated among the available resources:

- the usage of information about the Grid infrastructure based on static and dynamic data, collected from past executions and current status;
- the automatic analysis and characterization of the application to execute, in order to model its behavior; and,

- the employment of a newly created algorithm called *Dynamic Trapezoidal Self Scheduling* (DyTSS) that is able to determine the optimum size of every chunk in the moment of its execution, thus being able to adapt the submission of jobs to changes in the number, quality or size of the resources being used in a dynamic environment.

Hence, Montera first makes a profiling of the code to be executed in order to know its characteristics and the way that it works since MC applications follow a common schema of work: checking of input data, initializing data structures, etc.; simulation of the desired number of samples; and, recompilation of results and further final value. Because of this, just two parameters are necessary to define the magnitudes for the profiling; *constant effort* represents the necessary effort to execute the constant part of the application, and *sample effort* is the necessary effort to simulate a single sample.

Later on, an iterative mechanism implemented in Montera that takes the whetstone benchmark [9] as unit, measures the code performance on any remote site.

For the characterization of the Grid sites, as many variables as possible are considered: published information; whetstones; queue time; bandwidth; number of successful and failed tasks; etc. When necessary, not only the average value, but also the deviation is calculated.

This information is obtained with two complementary tools. The first time that a new site is discovered, Montera performs a profiling -based on whetstone benchmark and the copy of a 50MB file- in order to calculate its performance, bandwidth and queue time. Then, each time a task is submitted to a known site this information is dynamically updated: the application profile and execution time are employed to update the performance, its size to update the bandwidth, and the beginning of the execution to update the queue time. This way Montera increases its knowledge about the infrastructure whenever a task is successfully executed, thus being able to perform more accurate schedulings.

In addition, a series of rules are implemented inside Montera for profiting as much as possible of the free available slots in any time; thus, sites that fail in queuing jobs or over submission of the previous number of allocated/published jobs are banned for a fixed period of time, and the most reliable sites receive more tasks to execute.

The performance of the Grid infrastructure is modeled with the *asymptotic performance* and *half performance length* parameters [10], [11], but with the assumption of a variable task size.

For its part, the scheduling algorithm implemented in Montera is the so-called *Dynamic Trapezoid Self-Scheduler* (DyTSS) algorithm, which is an adaptation of *Grid Trapezoid Self-Scheduler* (GTSS) [12]. DyTSS is based on three elements: the performance analysis of each site and the whole Grid infrastructure; the profiling of the application to execute; and the GTSS dynamic algorithm. Employing this information, Montera calculates the ideal chunk size and number of chunks to execute on each site, with the sites having a higher performance, less queue time and higher bandwidth getting bigger chunks. A more detailed explanation about how DyTSS works can be found in [6], but an important concept must be mentioned: a dynamic approach is employed in the local scheduler by creating every chunk just before its submission -not all of them at the beginning of the execution- thus being able to adapt to changes in the Grid infrastructure.

The scheduling is done in a two-fold approach. As in previous schedulers, it is performed locally in order to determine the size of the chunk to submit to each remote resource prior to the proper execution; but as a new concept, it is also executed remotely. In this way, the chunk dynamically adjusts its size depending on the status of the remote resource and the user needs.

A. Storage Elements on Montera

In MC codes that do not require heavy data handling, their executable can be directly submitted and run without sending big files. That it is not the case of ISDEP since it must deal with the magnetic configuration of the device where it is being applied.

Thus, for the Large Helical Device (LHD) case, a binary file of around 7 MB representing such a configuration ought to be present on the Working Node prior to the execution of ISDEP. To do so, the original version of ISDEP downloaded it from a Storage Element. On the other side, the output file of that job represented no major problem for its retrieving since it weights for approximately 1 MB once it was remotely zipped.

This approach can also be followed with Montera, transparently to the framework. Given that a user-defined script controls the execution of the remote application, if it includes the retrieval of the desired input files the operation is automatically performed. This reduces the overhead related to the copy of the input data, and allows having files with a size that overcomes WMS limit. As a drawback, the employment of a Storage Element requires gLite calls (*lcg-cp*), restricting the execution of the application to sites where this middleware is available.

On the other side, GridWay -and, by extension, Montera- employs Globus Toolkit (*gsiftp*) to transfer the input files to the remote sites. It does not establish a file size limit, but obliges to copy the files to the remote site individually from the local resource.

Depending on the configuration of the remote sites and taking into account the aforementioned information, the final user has to decide which approach is more convenient to his objectives: a higher overhead in the submission stage, or a limitation on the set of sites to be used. In order to maximize

the instruments that can be employed by the final users, in this work both approaches were implemented.

IV. Results

Once that a statement of motives for using Montera applied to fusion Monte Carlo codes has been presented, the results related to ISDEP performance are described hereafter. Any of them corresponds to tests performed in July 2010.

A. Test-bed

A real Grid production infrastructure has been used, that supporting the *fusion* VO. It comes from the EGEE series of projects, which was at their end superseded by the EGI initiative, funded by the European VII Framework Programme.

When this work was carried out, *fusion* counted on 40 sites from 23 different organizations, with more than 25.000 CPUs (7300 of those were available at the moment this experiment was performed).

The local resource where Montera was executed consisted on a virtual machine running Debian 4.0, with GridWay 5.4.0 and Java Virtual Machine 1.5.0.09. This resource was in production status -being employed by different users to perform their daily job- with a quite restrictive configuration: a scheduling interval of 30 seconds, a dispatch chunk of 15 tasks, and a maximum number of simultaneous jobs per user of 100.

B. Performance

As aforementioned, ISDEP calculates the solution of the guiding centre equations in the presence of collisions with ions. In the particular use case employed in this work, the initial division of the problem has been done in jobs that integrate ten (10) trajectories for the LHD fusion reactor, i.e. a single job calculates that number of tasks. The execution of such a job lasts for around 8 hours in a single PC (Pentium IV@3GHz, 2Gb RAM).

A minimum and valuable physical research requires of fifty thousand trajectories (50.000) at least, so the reader can easily infer the vast amount of computational power needed for doing a former study. In this paper, results related to 500 jobs (5000 trajectories) are presented; such a number has been selected since it clearly represents the benefits for using ISDEP with Montera as a proof of concept without performing the whole research, the results and analysis of which are out of the scope of this paper. The way that such a whole physical research works, which is explained here for the readers convenience, is that thousands of identical jobs are submitted only differing on a random seed.

Table 1 shows the execution time when executing ISDEP on the *fusion* VO with 3 different tools: Workload Management System (WMS), a standalone GridWay [13] and Montera framework. In order to minimize the influence of external factors, the displayed values represent the average of three executions. As can be seen, GridWay experiments an improvement on the execution time of 11.8% compared with WMS, and Montera an additional 20.1% percent compared with the default scheduler of GridWay.

The inclusion of GridWay in this performance analysis is not intended to result on a deep comparison between GridWay and WMS, given that this has already been carried out in studies such as [14]. On the contrary, the intention is to be able to separate the proportion of improvement due to GridWay and the proportion due to Montera, in order to demonstrate the feasibility of the proposed approach.

Also, it is important to realize that WMS had a fault rate of 12.7%, while in the case of the two GridWay-based schedulers it is virtually zero. When employing WMS, the user has to be aware of any error that might happen during the execution and face it manually -for example, submitting the problematic task again-. GridWay incorporates an error-handling mechanism, so the task is automatically resubmitted until it finishes successfully. Montera does not have to face this problem because, as has been explained in the previous section, it does not depend on the correct completion of every single task.

C. Overhead

As in every experiment that involves the addition of hardware or software layers, the induced overhead must be measured. Here, this overhead comes from two updates: the substitution of WMS by GridWay, and the later addition of Montera.

As has be explained in the test-bed description, GridWay has been configured with a restrictive set of parameters. The establishment of a 30 second task scheduling interval adds an average overhead of 15 seconds per task. Anyway, given that the total execution time of this experiment was more than three thousand hours, it can be interfered that this does not significantly influence performance. The same happens with the 15-jobs per scheduling interval: although it can represent a bottleneck in applications composed by a massive amount of very short tasks, in this case its influence is negligible.

Montera itself is a very lightweight Java framework, whose execution length stays in the seconds time scale. The most significant overheads it induces are in the steps of resource and application profiling: a 300 second long task is submitted to each resource the first time it is discovered, and, to perform the application profiling, ten tasks with an execution time of about 10.000 seconds each are submitted in parallel.

Depending on the usage of Montera, the influence of these mechanisms can be very variable. Montera has been designed for a long term usage, so the first time it runs it spends a long time analyzing the infrastructure and application. If the intention of the final user is to perform just an execution of a given application, and GridWay is not available in the local resource, then WMS should be used. If GridWay

Table I
EXECUTION TIME AND FAULT RATE OF ISDEP WITH DIFFERENT SCHEDULERS

Scheduler	Walltime [hh:mm:ss]	Improvement (absolut) [hh:mm:ss]	Improvement (relative)	Fault Rate
fusion WMS	68:58:39	-	-	12.7%
GridWay	61:41:1	7:17:38	11.8 %	0%
Montera	51:5:56	17:52:42	34.98 %	-

is available, then it represents the best choice for small experiments, due to the performance increase and minor overhead. And if the users intention is to employ the Grid on a regular basis, the performance improvement obtained with Montera clearly overwhelms its long initialization time.

V. CONCLUSIONS

ISDEP is a highly computational demanding MC fusion code devoted to solve the plasma dynamics in a fusion device. It is undoubtedly a very promising tool to be used in the near future by the fusion community, so an efficient distributed execution is an asset. Because of this, Montera, a framework implemented to better execute MC codes on the Grid, can be a useful tool for better overcoming this challenge.

First tests, carried out on a subset of input trajectories extracted from a common ISDEP research and on the *fusion* VO, have showed an improvement on the executions of 11.8% when GridWay is employed instead of the default WMS scheduler, and an additional 20.1% with the usage of Montera. A bigger improvement is expected with the whole set of trajectories because, as it has been stated, short runs favor WMS and long runs favor Montera and its dynamic scheduling system.

Monte Carlo codes are widely employed in different areas of knowledge. It is worth mentioning success cases on radiological medicine [15] or economy [16], just to mention some. Montera has been designed with the purpose of seamlessly executing MC codes on the Grid. Thus, it is expected to be employed with an extensive set of applications. In this context, this work should not only be seen as the execution of a particular code in the Grid, but as the proof that Montera can deal with applications in a production status, execute them with a reduced effort from the user, and still obtain significant performance gains.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreements 211804 (EUFORIA) and 222667 (EGEE-III).

REFERENCES

[1] H. Wobig, "Theory of advanced stellarators," *Plasma Phys. Control. Fusion*, vol. 41, pp. A159–A173, 1999.

[2] J. P. Christiansen and J. W. Connor, "Ion transport from collisions and finite guiding centre drift excursions," *Plasma Phys. Control. Fusion*, vol. 46, pp. 1537–1566, 2004.

[3] F. Castejón *et al.*, "Ion kinetic transport in the presence of collisions and electric field in TJ-II ECRH plasmas," *Plasma Phys. Control. Fusion*, vol. 49, p. 753776, 2007.

[4] C. Alejaldre *et al.*, "TJ-II, a flexible heliac stellarator," *Fusion Technol.*, vol. 17, pp. 131–139, 1990.

[5] J. L. Velasco *et al.*, "Ion heating in transitions to CERC in the stellarator TJ-II," *Nucl. Fusion*, vol. 48, p. 065008, 2008.

[6] M. Rodríguez-Pascual, I. M. Llorente, and R. Mayo, "Montera: A framework for the efficient execution of Monte Carlo codes on grid infrastructures," *J. Parallel Distributed Computing*, in print, 2011.

[7] R. J. Goldston and P. H. Rutherford, *Introduction to Plasma Physics*. Bristol, England: IOP Publishing, 1995.

[8] T. S. Chen, "A general form of the coulomb scattering operators for Monte Carlo simulations and a note on the guiding center equations in different magnetic coordinate conventions," Max-Planck-Institut fur Plasmaphysik, Garching, Germany, Tech. Rep. 0/50, 1988.

[9] H. Curnow, "A synthetic benchmark," *Computer Journal*, vol. 19, pp. 43–49, 1976.

[10] R. W. Hockney and C. R. Jesshope, *Parallel Computers 2: Architecture, Programming and Algorithms*. Bristol, England: IOP Publishing Ltd., 1988.

[11] R. S. Montero, E. Huedo, and I. M. Llorente, "Benchmarking of high throughput computing applications on grid," *Parallel Computing*, vol. 32, pp. 267–279, 2006.

[12] J. Herrera, "Modelo de programación para infraestructuras grid computacionales," Ph.D. dissertation, Universidad Complutense de Madrid, Madrid, Spain, 2008.

[13] E. Huedo, R. S. Montero, and I. M. Llorente, "The GridWay framework for adaptive scheduling and execution on grids," *Scalable Computing-Practice and Experience*, vol. 6, pp. 1–8, 2005.

[14] J. Vázquez-Poletti, E. Huedo, R. S. Montero, and I. M. Llorente, "A comparison between two grid scheduling philosophies: EGEE WMS and GridWay," *Multiagent and Grid Systems*, vol. 3, no. 4, pp. 429–439, 2007.

[15] L. Maigne *et al.*, "Parallelization of monte carlo simulations and submission to a grid environment," vol. 14, pp. 177–196, 2004.

[16] H. Tanizaki, "Nonlinear and non-gaussian state-space modeling with monte carlo techniques: A survey and comparative study," Tech. Rep., 2000.

FAFNER2: A Comparison between the Grid and the MPI Versions of the Code

Manuel Rodríguez-Pascual, Francisco Castejón,
Antonio Juan Rubio-Montero, Rafael Mayo García
CIEMAT, Avda Complutense 22. 28040 MADRID (Spain)
manuel.rodriguez@ciemat.es

Ignacio Martín Llorente
DSA-Research.org, Universidad Complutense de Madrid,
C/ Prof. José García Santesmases s/n. 28040 MADRID (Spain)

ABSTRACT

FAFNER2 is a 3D code that simulates by Monte Carlo methods the Neutral Beam Injection (NBI) technology. The original version was implemented for shared memory computers with MIPS proccesors, so an update to be executed by means of MPI on standard Linux clusters has been carried out as well as a new version to be run on Grid. To do the latest, a serial version has also been developed, together with a Java DRMAA program that is submitted by the GridWay metascheduler. As a result, two new improved versions of the code (HPC and Grid) are available.

KEYWORDS: NBI; Grid; MPI; DRMAA

1. INTRODUCTION

Nuclear fusion is expected to be an energy source without the present environmental problems, socially acceptable and virtually inexhaustible. Several techniques for heating the plasma are used in the fusion experiments. Thus, the plasma can be heated with radiofrequency, using for instance waves in the Ion Cyclotron Range of Frequencies (ICRF), which heat ions, by means of the Electron Cyclotron Resonance Heating (ECRH) affecting (only) the electrons, or by the Neutral Beam Injection (NBI) heating.

The NBI method is widely employed to heat the plasma in fusion experimental devices. For the sake of evidence, it has been chosen as one of the heating methods for ITER [1]. The method is conceptually rather simple: neutral atoms are able to overcome the confining magnetic field of a tokamak or a stellarator and are ionized in the plasma via collisions with ions and electrons.

The main problem of nuclear fusion is technological, since nowadays it is not possible to build a reactor that could start and maintain the necessary fusion reactions for a long time. Computer simulations can help to solve this problem. There exist software applications that simulate different sections and phenomena of a fusion reactor. In this context, FAFNER [2] simulates NBI heating. It can be employed by its own or coupled to other codes, thus creating complex workflows that can simulate a wider range of phenomena inside the plasma.

The computational simulation for the optimization of an NBI heating mechanism is of outmost importance for the most possible efficient design, for example the threedimensional Monte Carlo (MC) shielding analysis on the ITER NBI duct for the nuclear and Bremsstrahlung radiation [3]. The only drawback is that this kind of simulation needs a lot of computing resources to produce valid results in a reasonable time period. To overcome this issue, multiple solutions have been adopted during the evolution of computational environments for scientific computing. This paper is focused on two alternatives, Grid Computing and Parallel Computing on a local cluster, as they are widely considered among the most successful platforms to execute parallel, loosely coupled applications.

Grid infrastructures can be used by the fusion community to share their equipment, creating a big pool of computational resources. This allows scientists to carry on wider,

more ambitious experiments, thus this alternative has arisen as a essential tool to many researchers. On the other side, the employment of local High Performance Computers rely on the resources of the researchers institution or the access to external ones, which is not always possible. They usually employ a front-end to communicate with the users, and rely on batch-queuing systems to efficiently distribute the tasks.

The code used in this work, FAFNER2, is the adapted version of FAFNER to the helical stellarator TJ-II [4]. It has been updated in order to be executed on modern clusters, and then modified to be efficiently executed on Grid infrastructures. To do so, a serial version of the code has been created, and then a DRMAA [5] application has been developed to manage the execution of multiple instances of the application in different remote *sites*. GridWay [6] metascheduler has been employed to monitor and control these executions.

2. THE FAFNER CODE

In nuclear fusion devices, one of the problems to solve is the creation of the extreme temperature and pressure required for the onset of nuclear fusion processes. In this context, NBI heating is an outstanding tool: by injecting energetic neutrals into the plasma both parameters are increased, thus reaching the necessary border conditions. The physical problem to solve then is the modelling of neutral beam injection into three-dimensional toroidal plasmas, and to calculate the trajectory of the resultant fast ions until they get lost or are absorbed by the system.

The FAFNER code simulates the injection of fast neutral particles into a toroidal plasma and the orbits of the resulting ions [2]. Thus, the global efficiency, the losses (i.e. shine-through, charge exchange and orbit losses, including those to limiters), the birth profile and the heating profile can be calculated for different discharges. For doing so, the input required data are the field configuration, the density and temperature profiles and the beam parameters. Instead of real space coordinates, in a second version of the FAFNER code the guiding centre part in magnetic coordinates is treated. The simulation is performed in two steps. The first one simulates the injection of the neutral beam into the torus, and obtains the coordinates and speed of the neutral particles. This information is employed in the second step to determinate the initial ionization of these atoms inside the plasma, as well as the energy distribution of the plasma after interacting with the fast ions. FAFNER2 also computes the energy lost as a result of the collisions between the neutral beam and the scrappers and ducts employed to shape the neutral beam before its injection to the torus.

From a mathematical point of view, MC techniques are employed to compute the coordinates of a set of fast ions, corresponding to a neutral beam, as well as the appropriate interaction of the beam with the background plasma. The trajectory of these ions and its interaction with the plasma are described in terms of a Fokker-Planck equation. The trajectories of the fast ions are solved with a Boozer flux coordinate system. FAFNER2 provides two sets of data on each execution. First, the position and speed of a given number of particles is determined. Then, it obtains different statistics about plasma heating process such as the fraction of ionized particles, absolute power provided to the torus, and fraction of lost particles.

3. GRID VS. CLUSTER COMPUTING

For the sake of completion, in this section Grid and cluster computing paradigms will be compared. The similarities and differences will be pointed out, thus helping to understand the motivation of this work. Its important to note that the definition of a cluster is standard and widely accepted among the scientific community, but this agreement is far from being achieved in the case of Grid computing. Here, the definition proposed by Foster [13] is considered.

The first difference among a cluster and a Grid is the resource administrator. While the cluster is usually managed by someone of the same organization as the user, in the case of the Grid the resources belong to different organizations, each one with different hardware, software, usage and security policies. This can constitute a drawback for the user, as the application to be executed must fit the the constrains of every site. On the other side, applications that run on clusters should be designed to exploit the most of the software already installed in the system and its specific hardware architecture. While in local clusters the physical resources are constant -this is, the number and kind of nodes do not vary along the time-, Grid infrastructures are highly dynamic. Unlike local clusters, which are considered as reliable, execution of tasks on the Grid is more conflictive, so the application must be designed to overcome these problems. This includes techniques such as checkpointing, job migration or performance monitoring.

As in Grid computing the resources are distributed around a wide area (usually, on the international scale) the communication among both the different tasks of a job and with the user is performed over the Internet. On the other side, in low latency clusters the different nodes are connected by a fast network, usually Gigabit Ethernet or Infiniband. This constrains the kind of problem that can be deployed

among Grid resources to independent bunch of jobs, because the bottleneck of executing a completely parallelized application (on nodes offered by different sites) is too big to be avoided. Also, there are in fact some Grid sites with MPI support, but at the moment their small size and usage complexity makes the use of MPI codes on the Grid scarce. Nevertheless, MPI sites can play a key role in the future, providing the necessary support for running applications which scalability factor is proportional to the number of non-coupled groups of parallel tasks that can be independently distributed.

At last, as the correct execution of each Grid task -including data movement- depends on the Internet status and configuration of both the local and remote resources, there is a much bigger probability of failure, which the user has to deal with.

4. IMPLEMENTATION

With FAFNER2 being a MC code, MPI is only employed to divide the simulation to be performed at the beginning of the execution among the different tasks, and collect the partial results from all the tasks after the simulations have been completed. In this section the process to create a serial DRMAA version, suitable to run on the Grid, is detailed.

4.1. Serial DRMAA Application

Distributed Resource Management Application API, DRMAA [5] is a high-level API specification for the submission and control of jobs to one or more sites within a Grid infrastructure. The decision of employing DRMAA to control the application is motivated by the provided capabilities for controlling the whole execution cycle of an arbitrary number of tasks: their creation, submission to the remote site, error control and results gathering can be fully automatized. DRMAA allows the user to implement distributed applications with an easy-to-use interface, hiding the low level details of the Grid infrastructure. Also, depending on the chosen scheduler or metascheduler, the same application can be executed on private resources (SGE, Condor) or the Grid (GridWay).

The substitution of MPI by DRMAA in FAFNER2 was accomplished in two steps. First, MPI was removed from the code, thus obtaining a serial application. Then, a Java DRMAA application was implemented, employing GridWay [6] to submit sequentially the serial version of FAFNER2 to the Grid and recover the results.

On the MPI version of FAFNER2, when the application starts, the first task checks the input data, initializes certain data structures and perform common calculus. Then, the simulation to perform is split among all the tasks. Being a MC code there is no need of interprocess communication, so each one is fully independent. After all the tasks have ended, the first one recovers the partial results and combine them to obtain the output of the application. To emulate this behavior, we firstly developed a serial version of FAFNER2 that behaves like the first MPI task at the beginning of the execution -reading the input data, initializing data structures and so- and then as an specific task of the MPI version, in order to perform the desired fraction of the simulation. This serial version shares the execution flow of the MPI one until the first call to $MPI\_BARRIER$, the point where the simulation is split among the different tasks, is done. Then, it behaves as one particular MPI task whose ID was received as an input parameter, instead of obtaining it from a MPI_COMM_RANK call.

When the different simulations have finished, the MPI version of FAFNER2 employs another $MPI\_BARRIER$ to synchronize them, gathers the partial results with MPI_GATHER and MPI_REDUCE calls and combines the data to obtain the final result. In the serial version these gathering operations are not needed anymore, so they have been removed. Instead, the partial results from each tasks are processed in the different Grid resources when possible, and sent back to the local site, since it is necessary to join all of them in order to obtain the final result.

In order to be able to merge the results of several FAFNER2 executions, the code had to be modified. FAFNER2 produces output data in two different places: along the program execution, and after performing all the calculations. This information has been processed when it is shown on the screen or stored in a file (for example, statistical analysis have been performed), so the application was modified to provide raw data. To achieve this target, the required variables and partial results are exported to an XLM file. These files, together with the program output, constitute the input of the post-process *wrapper*. This *wrapper* reads all the XML files, and together with the information of the output files generates the result of the whole simulation.

The next step consisted on the creation of a DRMAA application that employs the Grid to execute multiple, concurrent instances of our serial version of FAFNER2. This DRMAA application receives two parameters as an input data: the input data of FAFNER2, and the number of instances to be executed. With this information it submits the jobs to the Grid, waits until all jobs have ended, recovers the results from each one, and combines them to obtain the final result employing the aforementioned *wrapper*.

By employing DRMAA it is ensured that the code can be

coupled to different schedulers and metaschedulers. In this case, GridWay has been employed. GridWay is a metascheduler that highly simplifies Grid execution of jobs by automatically performing the steps involved in job submission: system selection, system preparation, submission, monitoring, migration and termination [6]. It provides some key features [14] required by FAFNER2:

- Straightforward DRMAA connection. After linking the DRMAA library of GridWay with the Java DRMAA application, the submission and control of the grid executions is delegated to GridWay. It choses the best node, submits the application, monitors the execution and recovers the results.
- Scheduling capabilities: GridWay employs a dynamic scheduling system. It can detect when a new machine has been added or removed from a cluster, and redistribute the work load. Also, it detects the services provided by any machine.
- Fault detection & recovery capabilities. Transparently to the end user, GridWay is able to detect and recover from any of the Grid elements failure, outage or saturation conditions.

4.2. Architectural Design

First, the previously described *Java Application* determines the number and size of the tasks to submit to the Grid in order to fit the user requirements. Then, it employs DRMAA interface to connect with *GridWay*, which will execute theses tasks. *GridWay* obtains information about the remote resources through the *Grid Information Services*, employing its *Information Manager*. After its internal scheduler has decided where to execute a certain task, it employs the *Transfer Manager* to copy the information to the *Computing Element* and the *Execution Manager* to carry out the proper execution. This *Computing Element* can be a simple server, so the execution is directly carried out, or a large multiprocessor system, such a cluster (or a shared memory platform). In the later, a queue system like SGE or PBS is employed to determine the *Worker Node* (or CPU) where an instance of FAFNER2 will be executed. At last, after the execution has finished, the output results travel in the opposite direction in order to reach the local resource.

5. EXPERIMENTAL RESULTS

In this Section, an analysis of the results of the execution of FAFNER2 on a Grid and a local cluster is provided. Many fusion codes rely on MC techniques, such as ASCOT [15], MOCADI [16], MNCP [17] or ISDEP [18], so it can be inferred that a wide community of users is spread around the fusion arena. To the date, these researchers have used local supercomputers to perform their calculations, and nowadays some of them are being aware of what Grid can offer. That is why some results on computational testbeds related to how a particular code can be run on Grid or HPC is of usefulness for achieving the best performance and, at the same time, results on the shorter time as, for example, can be seen in [19].

Table 1. Characteristics of the *Euler* Cluster

Element	Characteristics
CPUs	144 blades with 2 Xeon 5450 quadcore 3.0 GHz
Memory	2GB RAM/core
Network	Double Infiniband 4X DDR
Peak Performance	13.8 Tflops (linpack rpeak)

5.1. Testbed Description

This experiment was carried out employing two different resources, a cluster and a Grid infrastructure. *Euler* (see table 1 for details), is a low latency cluster that designates different queues to the jobs to be executed depending on their requirements. In the one employed for this work the maximum number of running jobs per user is limited to 104, thus establishing an upper bound to the degree of parallelization.

The Grid Infrastructure employed, *EELA-2* (see http://www.eu-eela.eu), consist of 30 resource centres with more than 3000 computing nodes and 700 TB of storage service. As explained in section 3 it is a dynamic infrastructure and the number and kind of resources vary in time. In this context, table 2 represents a snapshot of *EELA-2* performance on the moment this work was executed. It must be considered that on *EELA-2*, although there does not exist a maximum number of jobs to be submitted at the same time, in this experiment has been constrained to 20 per site -except in the case of *axon-g01.ieeta.pt*, limited to 10-, in order to avoid the saturation of the infrastructure. Given that when this experiment was performed seven different sites where employed, this restriction establishes an upper bound of 130 simultaneous jobs.

It is important to note that in the case of *Euler* each CPU slot is devoted to the execution of the desired job, while in the case of *EELA-2* that is not necessarily true, due of the existence of hyperthreading processors, different O.S. configurations and all the software layers implied in Grid monitoring and control. Thus, in this experiment a CPU usage monitor was included on each job, in order to obtain

Table 2. Performance of the *EELA-2* Grid Infrastructure (Nov. 2009)

Site	% employed CPU	Slots
axon-g01.ieeta.pt	69	4
ce-eela.ceta-ciemat.es	91	14
ce-eela.ciemat.es	91	7
celac.ula.ve	90	6
kuragua.uniandes.edu.co	97	3
ce-01.macc.unican.es	84	4
gridgate.cs.tcd.ie	85	20
gantt.cefet-rj.br	49	10

Where *Slots* stands for the number of execution units employed on each Grid site, usually corresponding to a virtual proccessor.

the real percentage of CPU devoted to FAFNER2.

5.2 Performance

Figure 1 depicts the performance of *Euler* and *EELA-2* under a heavy work load, composed of 1000 independent, serial tasks of equal size. In this way, it is possible to compare the performance and throughput of both infrastructures executing the very same simulation, thus pointing out the differences between these two computational paradigms in a production environment. For the sake of simplicity, in the case of *EELA-2*, *queue time* groups all the overheads related to the remote execution of jobs: GridWay scheduling time, remote queue time and copy of the input/output files to/from the remote site. In *Euler* cluster, *queue time* represents the time since each task was submitted to the local scheduler until its execution started.

First, it is noticeable that the execution time in *Euler* is roughly constant among all the tasks, while in *EELA-2* it is highly variable. This is due to the employment of an homogeneous computational environment in the case of the cluster, and heterogeneous in the Grid. The fact that the execution time in *Euler* is significantly smaller is due to the superiority of its resources.

It can also be observed that queue time in *Euler* is a step function, while *EELA-2* is lineal. The cause is related to the homogeneity of *Euler* and the limit of 104 jobs per user: the first 104 tasks are started at the same time and have a similar execution time, so they end at the same time and the resources employed are devoted to the next 104 until they are ended, and so on. As aforementioned, there is a limit of 130 jobs on on *EELA-2* infrastructure, but as the resources are dispersed and not homogeneous, each job starts in a

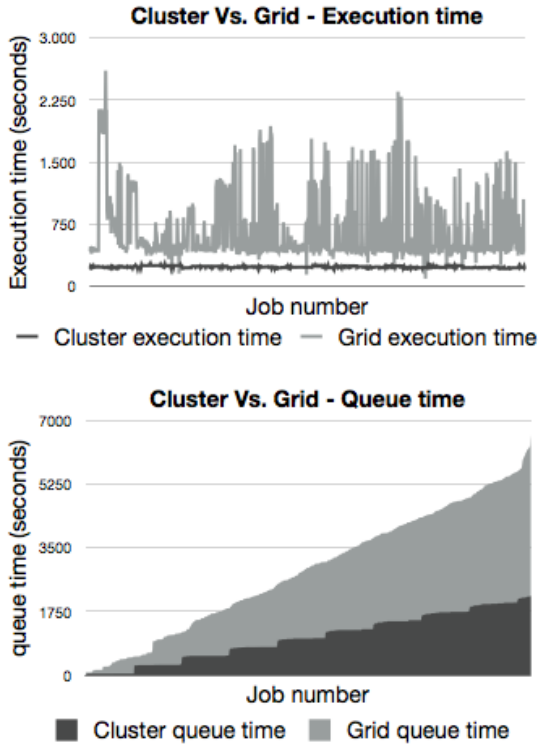

Figure 1. Queue and Execution Times in the *Euler* Cluster and the *EELA-2* Grid Infrastructure (1000 Independent Tasks).

different moment.

In the case of *EELA-2*, it is worth to notice that there is a certain number of jobs with a very high queue time, what greatly increases FAFNER2 wall time. This issue can be solved with a finer tune of the requirements and conditions detailed in the DRMAA template. A further analysis and the different alternatives to correct it employing GridWay can be found in [14].

5.3. Scalability

Figure 2 shows the scalability of the application in terms of number of particles with a single serial task. Here, two things must be taken in mind. First, the time of checking input data and generating output results is approximately constant -about 60 seconds- and that is why the results of 10 to 40 particles look identical; second, that the precision is one second, hence there results from the lower part of the graphic are not entirely accurate. As can be seen this scalability is lineal, what demonstrates that the application scales both in *EELA-2* and *Euler*. This is the expected result, as FAFNER2 is a MC code, and a bigger number of particles means a bigger number of iterations. This is a nec-

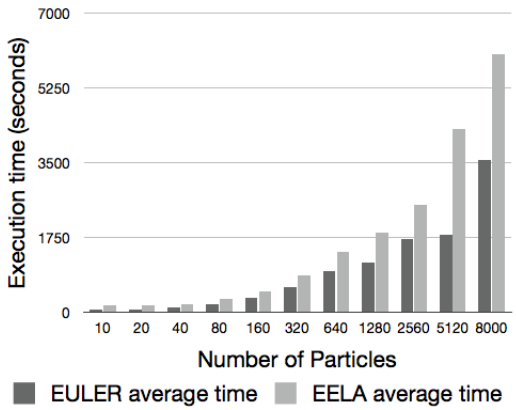

Figure 2. Execution Time in *Euler* and *EELA* With an Increasing Problem Size (1 node).

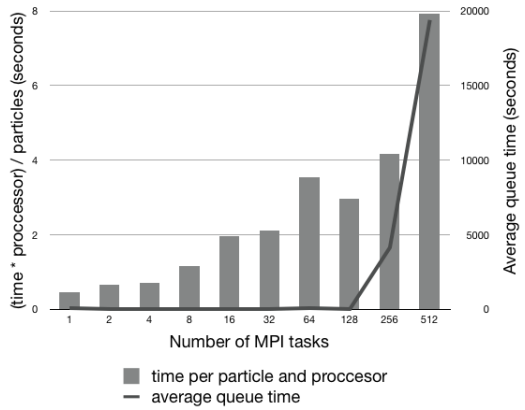

Figure 3. Execution Time in *Euler* With an Increasing Degree of Parallelism.

essary information in order to decide how to divide problem in subtasks. Depending on the available resources and time, the final user can decide whether to employ a bigger number of nodes simulating a small number of particles each, or the opposite. This fact provides the user an additional degree of freedom on the scheduling process.

If there is one key figure to understand the motivation of this work, that would be Figure 3: it shows that the scalability of the MPI version of FAFNER2 on *Euler* is limited in terms of number of nodes. Although the communication between tasks in FAFNER2 is very small, there do exist two barriers to synchronize them, placed at the beginning and near the end of the execution, so the application must wait to the slowest task to finish. The fact of not having an exclusive usage of the cluster, and the small performance differences among the tasks -inherent to the internal logic of the code- can become a bottleneck. The analytical data shows that when increasing the number of nodes from 4 to 512, execution time has only been divided by ten. Note that the number of available slots was increased to 512 for this particular test, in order to perform the scalability analysis.

In the case of the Grid version of FAFNER2 this issue is irrelevant, as the jobs to be executed are always serial. In this case the scalability is only limited by the resource availability. This allows the user to simulate an arbitrary number of particles in the desired number of nodes without any performance loss.

5.3. Overhead

Code serialization has an obvious problem. The operations and checkings that were executed only once in the MPI version, now must be executed in all tasks. This represents an overhead of approximately 60 seconds in our cluster *Euler*. The execution of FAFNER2 on remote sites also demands the copy of the input data and application executable, as well as the recovery of the partial results. These two overheads scale linearly with the number of nodes. They are both parallelizable: it is possible to transmit several files simultaneously without performance loss (depending on the network connection of both local and remote *sites*) and, once transferred, executions on the different *sites* are concurrent. Thus, their influence on the total execution time is very small.

The overheads produced in the local site must also be bore in mind. The first one is produced by the DRMAA application. Its execution time is about one second, so we consider it does not affect scalability. Also, GridWay has a scheduling interval of 30 seconds and a dispatch chunk of 15, i.e. 15 jobs dispatched every 30 seconds. This could in fact represent a scalability issue in problems composed by many small tasks, but is negligible with the existing problem scale.

The overhead produced by post-process script is more important. When executing FAFNER2 with a high degree of parallelization, this is a factor that must be taken into account, since it will increase linearly with the number of tasks. The final average is of two seconds/task.

In the case of the cluster version, Figure 3 also shows the relationship between the number of MPI tasks and the queue time, which also limits the scalability of the code. First, with *Euler* being a cluster with 1152 cores, relatively short tasks with a small number of nodes areeasy to allocate, and queue time is short. But lately, when the number of required slots is bigger, the job is harder to allocate and the queue time grows, sometimes up to the day scale. This is an aspect that final users should bear in mind in order to perform their experiments in a reasonable amount of time.

6. CONCLUSION

In this work, a comparative study among two of the most important paradigms of scientific computing has been accomplished. To do so, the same MC application has been employed in two production infrastructures. The results show that a cluster has a higher throughput in the case of small -in terms of parallelism- jobs. In this case, a Grid infrastructure also represents an alternative in the case that the final user could not have access to an HPC cluster or the use of this infrastructure is not allowed for embarrassingly parallel problems. In the case of loosely coupled jobs that can be ported to the Grid, the employment of serial applications plus a DRMAA controller instead of MPI removes all limitations to parallelism but the resource availability.

As a conclusion, both alternatives have their range of usage, depending on the computing needs for each run. Ideally the final user could execute small, fast experiments on his local cluster, and employ the Grid for larger experiments. This approach is specially good with FAFNER2 as it is not only employed to perform long simulations, but also smaller ones -several thousands of particles-.

REFERENCES

[1] N. Miyamoto *et al.* "Experimental results on ITER-NBI concept source," *AIP Conference Proceedings*, vol. 380, pp. 300–308, 1996.

[2] G. Lister, *FAFNER: A Fully 3-D Neutral Beam Injection Code Using Monte Carlo Methods*. Max-Planck-Institut für Plasmaphysik, 1985.

[3] S. Sato, H.Iida, M. Yamauchi, and T.Nishitani, "Shielding design of the ITER NBI duct for nuclear and bremsstrahlung radiation," *Radiation Protection Dosimetry*, vol. 116, no. 1,4, pp. 28–31, 2005.

[4] A.Teubel, J.Guasp, and M. Liniers, "Monte carlo simulations of NBI into the TJ-II helical axis stellarator," *Report IPP 4/268*, vol. 264, no. 4, 1994.

[5] A. H. Peter Troger, Hrabri Rajic and P. Domagalski, "Standardization of an API for distributed resource management systems," *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*, pp. 619–626, May 2007.

[6] E. Huedo, R. S. Montero, and I. M. Llorente, "The gridway framework for adaptative sheduling and execution on grids," *Scalable Computing – Practice and Experience*, vol. 6, no. 3, pp. 1–8, 2006.

[7] A. Boozer, "Guiding center drift equations," *Phys. Fluids*, vol. 23, pp. 904–909, 1980.

[8] ——, "Evaluation of the structure of ergodic fields," *Phys. Fluids*, vol. 5, no. 26, pp. 1288–1292, 1983.

[9] A. Teubel and F. Penningsfeld, "Influence of radial electric fields on the heating efficiency of neutral beam injection in the W7-AS stellarator," *Plasma Phys. Control. Fusion*, vol. 36, pp. 143–152, 1994.

[10] J. Guasp, M. Liniers, C. Fuentes, and G. Barrera, "Thermal load calculations at TJ-II vacuum vessel under neutral beam injection," *Fusion Technology*, vol. 35, pp. 32–41, 1999.

[11] J. Guasp and M. Liniers, "Loss cone structure for ions in the TJ-II helical axis stellarator. part I: properties without a radial electric field," *Nuclear Fusion*, vol. 40, p. 397, 2000.

[12] M. Gobbin *et al.*, "Numerical simulations of fast ion loss measurements induced by magnetic islands in the ASDEX upgrade tokamak," *Nuclear Fusion*, vol. 49, p. 095021, 2009.

[13] I. Foster, "What is the grid? a three point checklist," *Grid Today*, vol. 1, no. 6, 2002.

[14] ——, "Evaluating the reliability of computational grids from the end user's point of view," *Journal of Systems Architecture*, vol. 52, no. 12, pp. 727–736, 2006.

[15] J. Heikkinen, S. Sipila, and T. Pattikangas, "MPI-2: Extending the message-passing interface," *Comput. Phys. Commun*, no. 76, 1993.

[16] H. Mazzocco et al, "MOCADI_FUSION: Extension of the Monte-Carlo code MOCADI to heavy-ion fusion-evaporation reactions," *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, vol. 266, pp. 3467–3480, 2008.

[17] T. E. Booth *et a*l, "MCNP5 1.50 release notes," *Los Alamos National Laboratory report LA-UR-08-2300*.

[18] F. Castejón, L. A. Fernández, J. Guasp, V. Martin-Mayor, A. Tarancón, and J. L. Velasco, "Ion kinetic transport in the presence of collisions and electric field in TJ-II ECRH plasmas," *Plasma Phys. Control. Fusion*, no. 49, pp. 753–776, 2007.

[19] T. Desell *et al*, "Distributed and generic maximum likelihood evaluation," *Proc. Of the Third IEEE International Conference on e-Science and Grid Computing*, p. 8, 2007.

A Grid version of the Fusion code FAFNER

M. Rodríguez-Pascual[1], J. Guasp[2], F. Castejón[2], A. J. Rubio-Montero[1], I. M. Llorente[3] and R. Mayo[1]
[1] Information and Communication Technologies Division
CIEMAT
Madrid, Spain
rafael.mayo@ciemat.es
[2] Spanish National Laboratory of Fusion
EURATOM-CIEMAT Association
Madrid, Spain
[3] Distributed Systems Architecture Research Group
Universidad Complutense de Madrid
Madrid, Spain

***Abstract*—FAFNER is code developed by Lister which simulates by Monte Carlo methods the Neutral Beam Injection (NBI) technology, one of the most extended heating methods for fusion devices. To the date, FAFNER has been usually run at CIEMAT adapted to the TJ-II helical axis stellarator on shared memory Cray architecture machines. From this version, FAFNER has been ported to the Grid in the framework of the EGEE Project. At the same time, this work is the first step of a more ambitious work since the code can be now coupled to many others such as ion transport tools. In this paper, all these preliminary advances are described as well as the performance and portability gains obtained.**

Keywords-NBI heating, Grid applications

I. INTRODUCTION

Computer simulation has emerged as one of the most promising tools to study the fusion plasmas behavior and the optimization of the fusion devices prior to their construction. There are software applications that simulate different sections and processes of a nuclear reactor, obtaining very accurate results. This way, we can study the expected behavior of the reactor, thus predicting physical experiments. Nevertheless, one drawback is present, i.e. the requirements of this kind of software are highly computational demanding.

In this context, computational Grids represent a powerful tool for the fusion community, as its members are able to share their equipment and obtain a big pool of computational resources. This possibility is then complementary to the traditional supercomputers. Several international projects like EGEE [1], EELA-2 [2] or EUFORIA [3] provide the physical resources -both CPU time and storage- needed for efficient executions of fusion applications. Grid computing has already shown its capability to be used to solve several types of problems that appear in fusion plasma physics and that involve distributed calculation: Monte Carlo codes and parameter scan problems are the most suitable for this purpose, as they require high computation time and the problem can be divided in fully independent tasks, which are distributed among remote grid resources.

This work makes use of results produced by the European Commission funded Projects Enabling Grids for E-sciencE (EGEE-III, contract number INFSO-RI-222667) and E-science grid facility for Europe and Latin America (EELA-2, contract number INFRA-2007-223797) through the Seventh Framework Programme

The aim of this paper is to explain how FAFNER, an application that simulates NBI (Neutral Beam Injection) heating, was ported from a local cluster to the Grid, in order to take advantage of the aforementioned computational resources. NBI heating is a key factor on the development of fusion reactors. In fact, this technique will be employed at ITER [4], the next generation fusion reactor, where FAFNER is expected to have a relevant role in its design and simulation.

FAFNER original code was released by G.C. Lister, from Max-Planck institute, in 1985 [5]. The version employed in this work is an adaptation of FAFNER to the geometry of the TJ-II stellerator located at CIEMAT. It was developed by the Fusion department of CIEMAT, in collaboration with the original authors [6].

II. WHAT FAFNER SOLVES

The physical problem to solve is to model the neutral beam injection into three-dimensional toroidal plasmas, and to calculate the trajectory of the resultant fast ions until they get lost or are absorbed by the system.

The first part of the code simulates the injection of neutral particles into the torus. As a result, the coordinates and speed of these particles are obtained.

After that, in the second part of the code, this information is used to determinate the initial ionization of these atoms inside the plasma, and then, the energy distribution of the plasma as a result of its interaction with the fast ions. FAFNER also computes the energy lost as a result of the collisions between the neutral beam and the scrappers and ducts.

From a mathematical point of view, Monte Carlo techniques are employed to compute the coordinates of a set of fast ions, corresponding to a neutral beam, as well as the appropriate plasma parameters.

The trajectory of these ions and its interaction with the plasma are described in terms of a Fokker-Planck equation. The trajectories of the fast ions are solved with a Boozer flux coordinate system. These numerical techniques are well known and have been previously employed to describe heating of neutral beams both in tokamaks and stellarators fusion devices.

FAFNER provides two sets of data on each execution. First, the position and speed of a given number of particles is determined. Second, it obtains different statistics about plasma state: fraction of ionized particles, absolute power provided to the torus, and fraction of lost particles.

III. IMPLEMENTATION

The process of porting FAFNER to the Grid was accomplished in three steps.

The first one consisted of a transformation of the message passing mechanism, from SHMEM to MPI, inside a shared memory SGI computer. Then, we ported the application to x86 architecture. And at last, we executed this code on the Grid.

A. *Message passing mechanism transformation: from SHMEM to MPI*

As said, the first task was to change the message passing mechanism, from SHMEM to MPI. SHMEM is a message passing technology present in some SGI systems, but not on standard clusters. As our final goal is running FAFNER on the Grid, it is mandatory to use standard technologies as MPI is.

SHMEM library is based on MPI version 1, and in many cases their routines are equivalent. Thus, the first step was to find and replace these routines. Then, the application was optimized in order to avoid unnecessary inter-task communications and barriers, and to take advantage of all the possibilities that MPI provides. As a consequence, execution time was slightly reduced. The results are shown below.

As Input/Output is different on SHMEM and MPI, it was also necessary to adapt the code in order to have exactly the same output files. These files are employed as entry files in other applications, and any difference will make them miss-work.

B. *Architecture improvement: from MIPS to x86*

After porting the code to MPI, we migrated the application from its obsolete MIPS architecture to the x86 platform. The Operating System supported by the application was also updated, from IRIX64 6.5 to Scientific Linux 4.4.

The first step consisted of deleting obsolete and/or unnecessary packages, replacing them with newer versions. We also minimized the size of the executable by removing all the unused routines from the source code of these packages. In the case of NAG mathematical library, its optimization produced a size reduction of 99%.

Also, a small number of routines utilized in FAFNER do not belong to Fortran standard, but to a private library implemented for a certain architecture and operating system, present on the computers when the original version of the program was created, back in 1994. When migrating the application, these routines become unavailable or have a not very significant different functionality. Thus, it is necessary to locate these routines and implement them again, to make the application works exactly as expected.

Originally, FAFNER was designed to be executed on 64 bits processors. Nevertheless, as EGEE hardware and O.S. are 32 bits, it is necessary to make the application work with this word size. We had to keep the original data size both for integer and real variables, in order to obtain, as much as possible, the same data precision. This situation caused an unalignment of some variables, so we re-ordered its declarations, and then forced alignment by employing a specific flag on the compiler. The final results are equivalent with a deviation lower than 1%.

C. *Porting to the Grid*

Given that the basic idea behind the Grid is being able to launch multiple, concurrent executions of the application, it is necessary to divide the problem to be solved, in order to have one execution per section.

As aforementioned, FAFNER is a Monte Carlo code, so it was not necessary to modify the algorithm to subdivide the problem. Instead, in the Grid porting process we focused on dividing the input and joining the output data. In our approach, we implemented two wrappers: one that pre-processes the application input data and one that post-processes the output files (see Fig. 1).

FAFNER produces output data in two different places: along the program execution -only on the first thread, i.e. the first tasks of execution of several concurrently ones- and near the end of the execution, after gathering data from all threads. This information has been processed when it is shown on the screen or stored in a file (for example, if it is a mean, it has already been calculated), so we modified the application in order to get the raw data, before being processed.

To achieve this, we introduced into the code sentences that export to an XML file the required variables and partial results. This way, besides having a fully functional application, we produce results that are employed on the gridification. These files, together with the program output, constitute the entry of the post-process wrapper. This wrapper reads all the XML files, and together with the information of the output files (also in plain text format) generates a new output file.

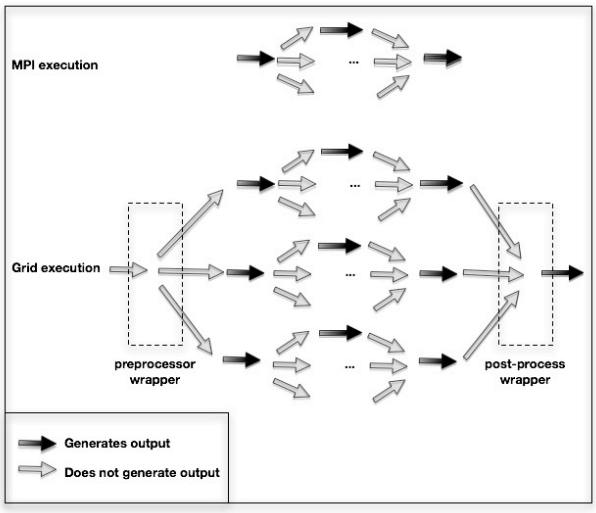

Figure 1. Generation of output files on MPI and Grid versions of FAFNER

To maximize the performance on the existing Grid infrastructure we decided to employ a metascheduler, in order to manage, control and monitor the execution of our Grid application. We have chosen the GridWay metascheduler [7], as it provides some features which are required by FAFNER:

- Scheduling capabilities: GridWay employs a dynamic scheduling system. It can detect when a new machine has been added or removed from a cluster, and redistribute the work load. Also, it detects the services published by any machine, so the application will only be sent to equipment running the correct MPI version.
- Fault detection & recovery capabilities. Transparently to the end user, GridWay is able to detect and recover from any of the Grid elements failure, outage or saturation conditions [8].

IV. Experimental Results

In this section we describe the results that we have obtained in this work. It is important to point out that in the porting process, the geometry of the fusion device (TJ-II in our case) was extracted from the code and set it up as an independent module which is called from the main program, so the code can now easily be run for any other stellarator or tokamak by simply changing this module.

A. Testbed Description

The behavior of the successive FAFNER implementations has been analyzed in a test-bed based on three resources: one shared memory computer and two MPI-enabled clusters. The main characteristics of these machines are described in Table 1.

TABLE 1. CHARACTERISTICS OF THE TESTBED RESOURCES ON JANUARY 2009

Name	***Resources characteristics***
Jen50, 122 cores (SGI Origin 3800)	MIPS R14000@500 MHz, 126 GB shared memory
Lince, 88 nodes - 290 slots (heterogeneous x86 cluster)	Dual Xeon 3.2 GHz,2 GB each 1Gb/s Ethernet
ce-eela.ciemat.es, 20 nodes - 120 cores (heterogeneous x86 cluster)	Dual Xeon 3.20 GHz 2 GB each 1Gb/s Ethernet

We also submitted jobs to different remote Grid nodes (sites) in order to measure file transfer time. We employed three different Grid nodes, located at different distances from CIEMAT:

- a very close one at Universidad Complutense de Madrid (UCM);
- a 325 Km far one at Universidad Politécnica de Valencia (UPV); and,
- a 8000 Km far one at Universidade Federal do Rio de Janeiro (UFRJ).

Table 2 summarizes the transfer time for the input and output files in an 8000 particles case, the reliable use case employed in this work.

TABLE 2. TRANSFER TIME FOR A 8000 PARTICLES FAFNER CASE

Grid node	***Input file (50 MB)***	***Output file (17 MB)***
aquila.dacya.upm.es	9.16 s	3.28 s
ramses.dsic.upv.es	60.8 s	19.3 s
ce-biof.eela.ufrj.br	438.75 s	151 s

In our case, the input file contains the FAFNER executable, the TJ-II geometry libraries and some additional information about the case of study; all of this usually sums up to 50 MB. On the other hand, the size of the output file depends on the required simulation and precision; in the shown example, it was of 17 MB.

B. SHMEM to MPI

First, we studied the performance gain obtained after upgrading the message passing mechanism. Both versions of FAFNER were executed 10 times on the same computer under different load conditions, to have realistic results.

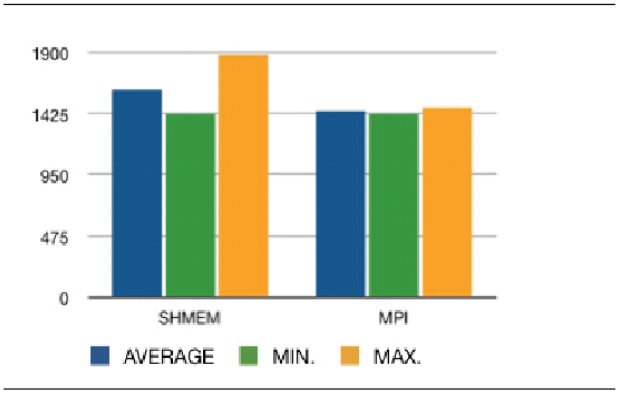

Figure 2. FAFNER execution time of SHMEM and MPI versions in Jen50

Fig. 2 shows these execution times. SHMEM usually outperforms MPI on MIPS processors [9]. However, after migrating the application and minimizing its network traffic, we have reduced execution time in 149 seconds, roughly a 10%. Also, time variability has been reduced: the difference between the fastest and slowest execution has been reduced from 458 to 51 seconds. This improvement has been produced due to the reduction of communication processes.

C. SGI to x86

After migrating the application from MIPS to x86, we compared its performance in both platforms. FAFNER will be executed on a local cluster when the phenomena to be simulated will calculate a low number of particles (some thousands).

Due to system use policies usually implemented on a production computing environment, MPI applications executed on *Lince* cluster are limited to 10 slots (this is, 10 jobs on a common submitted time), so results shown on Fig. 3 reports an improvement ratio of three.

D. *Cluster to Grid*

After porting the application to the grid, we measured its execution time. This allowed us to check, besides the execution time difference, the scalability of the problem. Grid execution is not only intended to reduce execution time of jobs, but to allow simulating more complicated phenomena, that cannot be coped with local infrastructures.

Measuring Grid performance is complicated. Depending on the chosen parameters, sites and moment on which the application will be run, results can be tremendously variables. To minimize this problem, we only employed an EELA grid node at CIEMAT. This way, we also had the possibility of monitoring system load, and time executions under different conditions -as we did on the previous cases.

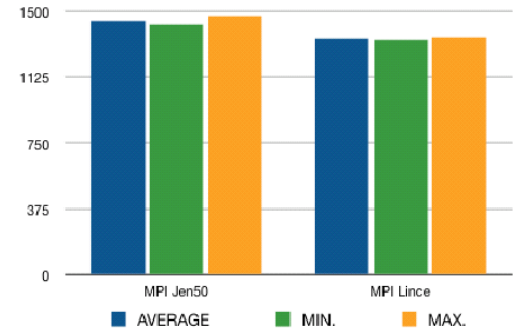

Figure 3. FAFNER execution time on Jen50 (32 threads) and Lince (10 threads).

In this case, we need to time not only the application execution, but also the wrappers execution time. These are the obtained results:

- Pre-process wrapper is relatively simple, and its execution very fast. Execution time is less than a second.
- Execution time of FAFNER was measured on the EELA-2 Grid node of CIEMAT, called *ce-eela.ciemat.es*. Obtained results are shown on Fig. 4. It is a key factor to bear in mind that for Acceptable Use Policy reasons in the *Lince* cluster, this figure don't show a real parametric comparison because we couldn't perform the executions on the same number of nodes; thus, we could count on 32 nodes for the *ce-eela.ciemat* case, but only 10 for the *Lince* one. These are also the number of MPI instances executed.
- Post-process wrapper scales linearly with the number of executions, at about 2 seconds/execution. In this case, with 10 executions, it took about 20 seconds to generate the final result.

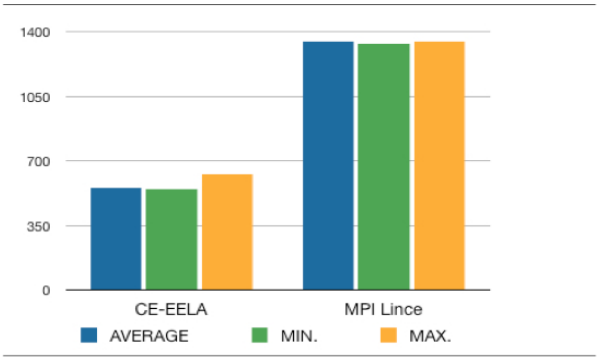

Figure 4. FAFNER execution time on *Lince* and *ce-eela.ciemat.es*

E. *NBI results*

For obtaining the previous statistics, real NBI cases belonging to the TJ-II device have been used. In Fig. 5 the reader can find a plot of three different magnitudes as a function of the line density: the percentage of particles which had a charge exchange in the heating process; the total power supplied to the toroidal plasma; and, the percentage of particles loosed into the plasma. These results were obtained with an 8000 particles case.

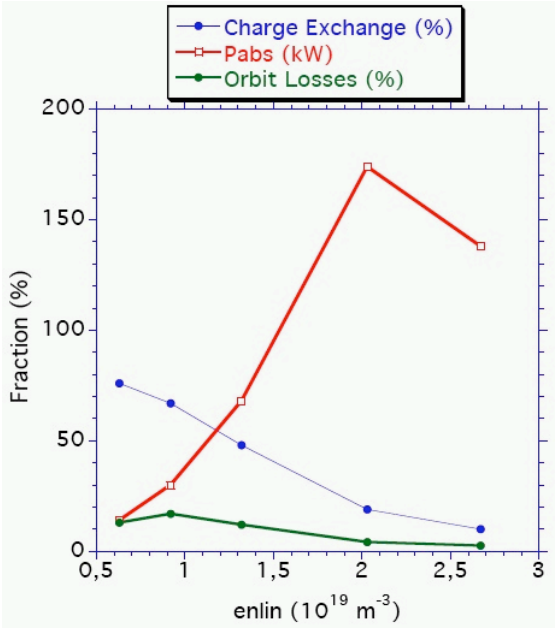

Figure 5. Evolution of some TJ-II plasma magnitudes provided by FAFNER

V. FUTURE WORK

FAFNER is an application that will evolve on the following years. Now it is able to cope with several thousands of particles in a valid use case (8000 in this work), but it can be increased in more than one order of magnitude with no problems. Besides being employed at CIEMAT to simulate the processes inside TJ-II, both on Grid and on MPI clusters, it is expected to simulate NBI heating at ITER [4], the biggest fusion project in the history of mankind. To perform this task, FAFNER code must be adapted, providing a clear and easy-to-use interface. As a consequence, the geometry of the sector

being simulated will be a different module in the code, so the user will be able to upload it as required.

In order to obtain a greater portability, MPI will be removed from the code, thus making FAFNER work on any Grid site. This way, the number of machines in which FAFNER can run will be greatly increased. The drawback is the need of implementing more complex pre-processor and post-processor wrappers.

At the same time, it is necessary to analyze FAFNER information flow and minimize it, in order to reduce the data being transmitted on the internet. This way execution time is reduced, and the exposure to a data transmission fault is minimized. To accomplish this task, it is necessary to implement a workflow, studying which data to send where and when.

When all this tasks are accomplished, a web interface to manage FAFNER and analyze its results will be created, in order to simplify the usage of the application. This interface will also allow coupling FAFNER to other fusion codes such as ISDEP [10], for example, creating complex workflows that simulate a wide number of plasma phenomena.

VI. CONCLUSIONS

During this work we have accomplished four fundamental objectives: a change on the message passing paradigm, from SHMEM to MPI; the update of the application to a newer architecture, from MIPS to x86; the porting of the application to the grid; and, the possibility of executing the code on any fusion device by changing an independent module representing its geometry. All of them are part of a more ambitious plan to be implemented afterwards.

The first task, changing the message passing interface, although being a long task was successfully solved, making the application work correctly.

The second task (porting the application from MIPS from x86) led to deep changes in the code in order to obtain the same results. Anyway, we managed to reduce execution time to about one third, maintaining data precision and the same output format.

Lately, we checked that FAFNER correctly works on a Grid. We developed pre-process and post-process wrappers. Then, we measured the performance of the application, and demonstrated its scalability.

At last, the geometry of the helical axis TJ-II was extracted from the code making its exchange an easier task.

The final result of the work is to have developed a NBI fusion code on the Grid, the use of which will be enhanced in a near future by coupling it to other applications in order to simulate the dynamics of fusion plasmas in a wider temporal and physical evolution. At the same time, new improvements in the code will be accomplished as was stated in previous sections.

REFERENCES

[1] The EGEE Project, available at http://public.eu-egee.org.

[2] The EELA-2 Project, available at http://www.eu-eela.eu.

[3] The EUFORIA Project, available at http://www.euforia-project.eu.

[4] T. Oikawa *et al.*, "An assessment of ITER scenarios under varying assumptions of NBI capability and NBCD code benchmark," in 34th EPS Conference on Plasma Phys. Warsaw, vol. 31F, pp. P-4.163, July 2007.

[5] G. Lister, "FAFNER: A Fully 3-D Neutral Beam Injection Code Using Monte Carlo Methods," in Max-Planck-Institut für Plasmaphysik Technical Report IPP 4/222, 1985.

[6] J. A. Teubel, J. Guasp, and M. Liniers, "Monte Carlo simulations of NBI into the TJ-II helical axis stellarator,".Report IPP 4/268, 264(4), 1994.

[7] E. Huedo, R. S. Montero, and I. M. Llorente. "The GridWay Framework for Adaptive Scheduling and Execution on Grids," Journal of Scalable Computing–Practice and Experience, vol. 6, pp. 1-8, 2005.

[8] E. Huedo, R. S. Montero, and I. M. Llorente, "Evaluating the Reliability of Computational Grids from the End User's Point of View," Journal of Systems Architecture, vol. 52, pp. 727–736, 2006.

[9] G. R. Luecke, S. Spanoyannis, and M. Kraeva. "The performance and scalability of SHMEM and MPI-2 one-sided routines on a SGI Origin 2000 and a Cray T3E-600," Concurrency and Computation: Practice & Experience., vol. 16, pp. 1037-1060, 2004.

[10] J. L. Velasco *et al.*, "Ion heating in transitions to CERC in the stellarator TJ-II," Nuclear Fusion, vol. 48, pp. 065008, 2008.

Proceedings of the Second EELA-2 Conference
R. Mayo et al. (Eds.)
CIEMAT 2009
© 2009 The authors. All rights reserved

FAFNER2: Executions of a code for estimating NBI heating of fusion plasmas on Grid

M.A. Rodríguez-Pascual[1], J. Guasp[1,2], F.Castejón[1,2], I.M. Llorente[3] and R. Mayo[1]

[1]CIEMAT, Avda. Complutense, 22 - 28040 Madrid (Spain)
{manuel.rodriguez,antonio.rubio,rafael.mayo}@ciemat.es
[2] Laboratorio Nacional de Fusión, Avda. Complutense, 22 - 28040 Madrid (Spain)
{guasp,francisco.castejon}@ciemat.es
[3] Universidad Complutense de Madrid. Avda. Ramón Santesmases S/N 28040 Madrid (Spain)
llorente@dacya.ucm.es

Abstract

FAFNER2 is a 3D code that simulates by Monte Carlo techniques the Neutral Beam Injection (NBI) technology, one of the most used heating method for fusion devices. It is adapted to the TJ-II helical axis stellarator from the original one developed by Lister at Max Planck IPP, but can easily be executed for other geometries by changing the independent associated module. In a first step, FAFNER2 was adapted to be run on the current architectures and paradigms (x86, MPI & DRMAA) also improving the old efficiency on shared memory Cray architectures. Nowadays, it can be executed on Grid and supercomputing platforms, but a real test on a production infrastructure was remaining. This work intends to overcome this issue by performing a massive test, the computational results of which will be a key factor for the final users in order to plan their calculus in a two-fold way: time spent and required accuracy in the physical values obtained.

1. Introduction

Fusion is the fundamental energy source of the universe. It is the process that powers the Sun and the Stars. In a fusion reaction, large amounts of energy are released when the nuclei of two light atoms (deuterium and tritium) fuse together to form a heavier one, helium. Tapping into this energy source offers the prospect of a long-term, safe, environmentally friendly option to meet the energy needs of a growing world population.

Fusion is a particularly attractive energy solution as it uses a fuel that is abundant and available everywhere. The primary fuels used in fusion are deuterium and lithium. Deuterium is a hydrogen isotope, which can be readily extracted from water (there is around 30g of deuterium in every cubic metre of water), and lithium is an abundant light metal from which tritium can be generated inside the reactor.

In hydrogen atoms the centre, or nucleus, contains only one proton. In deuterium the nucleus contains a proton and one neutron, while for tritium there are two neutrons with the proton. The fusion of one deuterium nucleus with a tritium nucleus makes a new nucleus of the element helium (also known as an alpha particle), a neutron and a huge amount of energy. The extra neutron can be used to generate more tritium fuel from lithium. One gram of fusion

fuel could generate 100 000 kilowatt hours of electricity, which is equivalent to burn eight tonnes of coal.

Fusion reactions occur at high temperatures when the nuclei collide with sufficient energy to overcome the natural repulsive forces of their electrical charges. They occur naturally in the sun at temperatures of the order of 10^6 Celsius degrees, producing the energy that sustains life on Earth. However, in the Sun the fusion fuel is heated and compressed by massive gravitational forces. On Earth we cannot use gravity, so the challenge for fusion researchers is to compensate by heating a lower-density plasma to a higher temperature (about 108 Celsius degrees, i.e. 10 times hotter than the core of the Sun) with excellent thermal insulation to initiate self-sustaining fusion reactions. This high temperature is well above the temperature at which a gas is completely ionised and becomes a plasma, the fourth state of matter. In an ionised plasma the positively-charged nuclei and negatively-charged electrons of atoms are separated and move about freely like molecules in a gas. More than 99% of our universe exists as plasma, but most of it at much lower temperatures.

To reach fusion conditions in the plasma, powerful heating is necessary and heat loss must be kept to a minimum by keeping the hot plasma thermally insulated from the reactor walls, a process known as confinement. This is a difficult task, both in terms of understanding the complex physical processes involved and developing the sophisticated technologies required to control them. Two different technologies have been developed in fusion research: magnetic confinement and inertial confinement.

Magnetic confinement uses strong magnetic fields to provide the thermal insulation of the plasma and allows the possibility of steady state operation, whilst inertial confinement uses high-power lasers or ion beams to heat and compress minuscule pellets of fuel.

The main problem of nuclear fusion is then a technological one: we still are not able to build a reactor that can start and maintain this kind of reaction for a long time. To overcome this problem, computer simulation is employed. There are software applications that simulate different sections of a nuclear reactor, obtaining very accurate results. This way, we can study the expected behaviour of the reactor, thus reducing physical experiments. The only drawback is that this kind of software needs a lot of computing resources to produce valid results in a reasonable time period.

In this context, computational grids represent a powerful tool for the fusion community, as its members are able to share their equipment and obtain a big pool of computational resources. Several international projects like EGEE [1], EELA-2 [2] or EUFORIA [3] provide the physical resources -both CPU time and storage- needed for efficient executions of fusion applications.

FAFNER original code [4] was released by G. Lister, from Max-Planck institute, for simulating the Neutral Beam Injection (NBI) heating technique, one of the most widely used worldwide in fusion devices. Lately, a modified version adapted to the geometry of the TJ-II stellerator located at CIEMAT was implemented by the fusion department of CIEMAT in collaboration with the original authors [5].

2. The code and its Grid version

A detailed explanation of what the FAFNER2 codes simulates and the way that it was ported to the Grid can be found in [6]; nevertheless, we here summary for the reader convenience.

The physical problem to solve is to model the neutral beam injection into three-dimensional toroidal plasmas, and to calculate the trajectory of the resultant fast ions until they get lost or are absorbed by the system.

The first part of the code simulates the injection of neutral particles into the torus. As a result, the coordinates and speed of these particles are obtained. After that, in the second part, this information is used to determinate the initial ionization of these atoms inside the plasma, and then, the energy distribution of the plasma as a result of its interaction with the fast ions. FAFNER2 also computes the energy lost as a result of the collisions between the neutral beam and the scrappers and ducts.

From a mathematical point of view, Monte Carlo techniques are employed to compute the coordinates of a set of fast ions, corresponding to a neutral beam, as well as the appropriate plasma parameters.

The process of porting FAFNER2 to the Grid was accomplished in three steps.

The first one consisted of a transformation of the message passing mechanism, from SHMEM to MPI. SHMEM library is based on MPI version 1, and in many cases their routines are equivalent. Thus, the first step was to find and replace these routines. Then, the application was optimized in order to avoid unnecessary inter-thread communications and barriers, and to take advantage of all the possibilities that MPI provides. As a consequence, execution time was slightly reduced.

Then, we ported the application to X86 architecture by deleting obsolete and/or unnecessary packages, replacing them with newer versions. We also minimized the size of the executable by removing all the unused routines from the source code of these packages. Also, a small number of routines utilized in FAFNER2 do not belong to Fortran standard, so they were implemented again on this basis, and the code was adapted to 32 bits.

And at last, we executed this code on the Grid. Since FAFNER2 is a Monte Carlo code, it was not necessary to modify the algorithm to subdivide the problem. Instead, in the grid porting process we focused on dividing the input and joining the output data. The code produces output data in two different places: along the program execution -only on the first thread- and near the end of the execution, after gathering data from all threads. This information has been processed when it is shown on the screen or stored in a file, so we modified the application in order to get the raw data, before being processed.

Last, a new version without MPI libraries, but DRMAA was implemented in order to execute the Grid version in any site. This release is able to submit the jobs by means of the GridWay [7] metascheduler.

3. FAFNER2 executions on Grid: a production test

In this section, an ambitious test executed on a Grid e-Infrastructure and the results it has provided will be described. In this sense, a detailed analysis on the physical obtained values with respect to the number of simulated particles has been done. Thus, bearing in mind to always have a right result, the final objective has been to find an upper limit of this number of particles to be calculated depending on the desired accuracy, evaluating in this way the suitability of the gridified FAFNER2 version on a production environment. This is of outmost importance for the final user in order to know the lowest cost-effective computation needed to find correct values.

A description of the different executions of the FAFNER2 code on the Grid e-Infrastructure is as follows in terms of resource availability, reliability and performance.

To maximize the performance of FAFNER2 on the Grid infrastructure, strict criteria about site performance has been set. By monitoring variables as execution wall-time, queue waiting time and CPU usage, jointly with an aggressive migration criterion, a high performance is expected. We have deeply analyzed the relationship between these performance enhancements and the overhead induced by the job cancellation or migration in sub-optimal sites. At last, we have tested different scheduling policies in order to find the most convenient

one depending on the user requirements–degree of parallelism, number of particles to simulate and deadline for a given FAFNER2 execution.

In all cases we have employed GridWay [7] metascheduler to control job submission and execution.

3.1. Analysis on the physical results obtained

The main target of this work consisted on testing whether the Grid implementation of FAFNER2 was in fact suitable to run on a Grid infrastructure. To do so, this experiment was done in two folds.

First, an increasing number of tasks with the same number of particles to be simulated were submitted to the Grid. Figure 1 shows the obtained physical results of one FAFNER2 output value: the proportion of total/lost particles. As can be seen, as the number of tasks increases, the final proportion of total/lost particles reach a threshold that can be taken into account by the user in his/her future executions. For the sake of comparison and estimation of the required accuracy by the user too, an additional value related to the Student's-t deviation is also plotted.

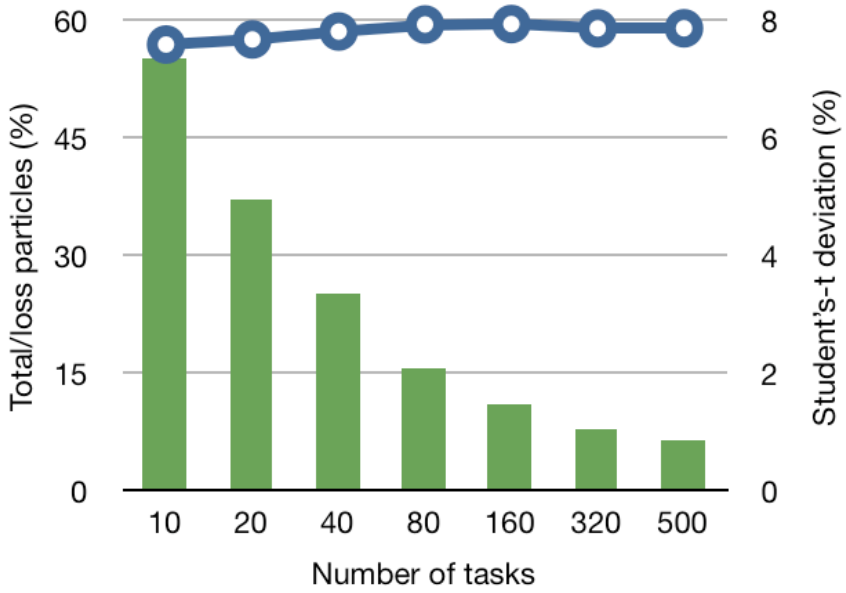

Figure 1. Evolution of the FAFNER2 parameter 'Proportion of total/lost particles' with the number of tasks submitted (blue line). The accuracy of the results by means of the Student's-t value is also depicted (green)

A second parameter is also analyzed in Figure 2. As can be seen, the value related to the Proportion of total/lost particles by FAFNER2 is used again, but with respect to the number of particles calculated per task. For this case, the total number of particles remains constant to 3200, so submitting 10 tasks corresponds to 320 particles per task and so on. With this test, a demonstration of how the application can be executed on the Grid with an increasing value of parallelism is proved. In this case, it is noticeable that the typical deviation is reduced when the size of the chunk is bigger, and Student's-t deviation increases in a similar proportion. This is due to the nature of the application: FAFNER2 returns the average value of the aforementioned parameter, and then we calculate both deviations. Increasing the number of particles on each task means that the average value returned will be more precise, so the typical deviation is smaller, but having an inferior number of tasks increases Student's-t deviation. Thus, for a final evaluation, both parameters should be taken into account.

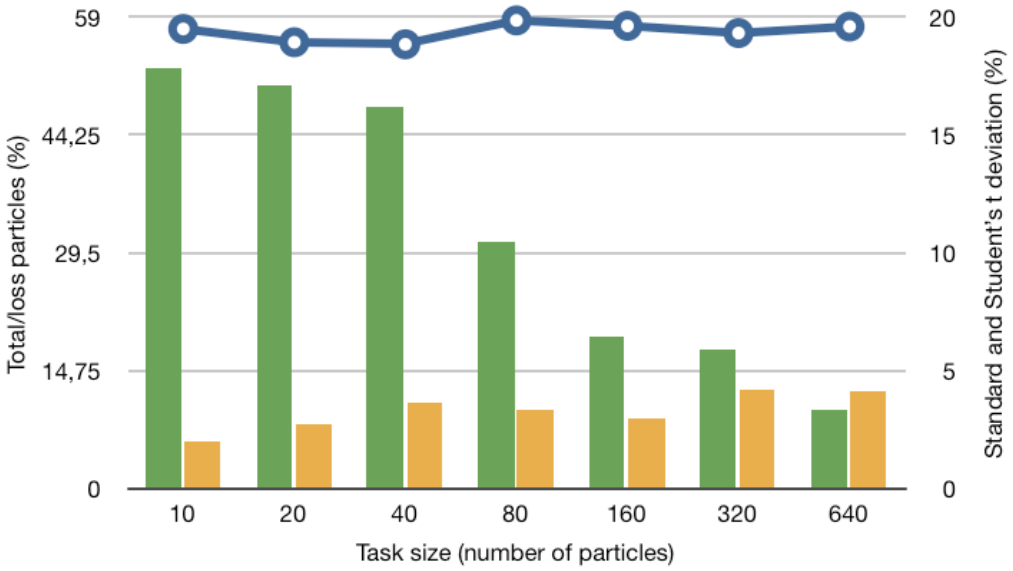

Figure 2. Evolution of the FAFNER2 parameter 'Proportion of total/lost particles' with the size of the tasks submitted (blue line). The accuracy of the results by means of the Student's-t value (orange) and standard deviation (green) are also depicted.

3.2 Analysis on FAFNER2 scalability

To study the scalability of our approach in terms of number of particles we employed the EELA-2 Grid infrastructure, in order to have the most accurate results possible regarding a particular environment. Results (see Figure 3) show that the execution time and the number of particles grow roughly in the same proportion.

Here, two things must be bear in mind. First, the time of checking input data and generating output results is approximately constant, and that is why the results start at about 270 seconds. Also, the size of the input and output files is exactly the same, which also contributes to a flatter graphic.

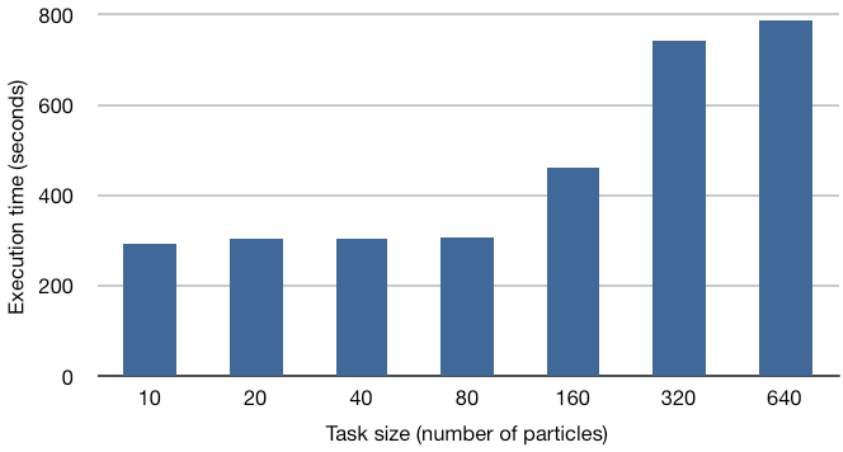

Figure 3. Total execution time (seconds), including data transmission time.

3.3 Analysis on the EELA-2 Grid Infrastructure

As the main part of this work was accomplished on the EELA-2 Grid infrastructure, we think it is important to show its performance under the computing demand of a heavy-user.

Our intention here is not to carry out a deep analysis of the infrastructure, but to show a "snapshot" of its performance in a given moment.

Site	%CPU	Nodes
ieeta.pt	69	4
ce-eela.ceta-ciemat ciemat,es	91	14
ce-eela.ciemat,es	91	7
celac.ula.ve	90	6
kuragua.uniandes.edu.co	97	3
ce-01.macc.unican.es	84	4
gridgate.cs.tcd.ie	85	20
gantt.cefet-rj.br	49	10
ieeta.pt	69	4

Table 1: performance of the Grid infrstructure

Table 1 shows the sites where we have carried out our experiment. To obtain the percentage of CPU devoted to our tasks, we created a small monitor that was submitted together with our application and was in charge of recording this CPU percentage in a text file. The average queue time was 156 seconds, with a standard/Student's deviation of 256/22 seconds. The whole test took 355 tasks successfully completed and only 23 were failed.

3.4 Advanced metascheduling configuration

As was mentioned at the beginning of this section, GridWay metascheduler [7] was employed to control the submission and monitoring of the Grid execution as it provides several tools to optimize the Grid performance of a given application depending on the status of the infrastructure.

Regarding the high proportion of queue/execution time, we have decided to set very restrictive migration conditions, which is of importance when running small tasks. As this proportion can grow up to a 30% of the total time, performing a task migration usually leads to a higher walltime -specially when the performance loss happens near the end of the execution-.

To avoid long queue waiting time, we have decided to set the "`SUSPENSION_TIMEOUT`" parameter to 300 seconds, roughly the double of the average queue waiting time. This way we can avoid saturated resources and focus our executions on the fastest ones.

In case the user had a restrictive deadline to finish the execution, Figure 3 provides a reference on the execution time depending on the task size. It is straightforward to obtain the lowest degree of parallelism necessary to meet his requirements. As it has been also demonstrated (Figures 1 and 2) that a higher degree of parallelism does not affect the quality of the results, the only drawback of this solution is an increasing overhead.

In order to analyze the reliability of the different sites, a reschedule of the task submission after a failure was not enabled. Anyway, in production environments this GridWay feature can be easily be activated, setting "`RESCHEDULE_ON_FAILURE=yes`". This allows the metascheduler to submit the task to any non-problematic site transparently to the final user, thus we can ensure the reliability of the proposed solution.

4. Conclusions

During this work we have accomplished a fundamental objective, i.e. to test the real performance of the FAFNER2 code on a production Grid infrastructure.

Since it is planned to be coupled to other fusion codes in a near future by means of more complex workflows, it is very important to test how the application works in order to face this new challenges successfully.

At the same time, an analysis on how the final physical results vary with the number of simulated particles has been done. With this information, the final user can weigh up the required accuracy *vs* computational time spent.

Acknowledgements

This work makes use of results produced on the EELA-2 project (http://www.eu-eela.eu), co-funded by the European Commission within its Seventh Framework Programme. The authors thank the consortium.

References

[1] http://public.eu-egee.org.

[2] http://www.eu-eela.eu.

[3] http://www.euforia-project.eu.

[4] G. Lister. FAFNER: A Fully 3-D Neutral Beam Injection Code Using Monte Carlo Methods. Max-Planck-Institut für Plasmaphysik, 1985.

[5] J.A. Teubel. Monte Carlo simulations of NBI into the TJ-II helical axis stellarator. Report IPP 4/268, 264(4), 1994.

[6] M. Rodriguez-Pascual *et al*. FAFNER-2: adaptation of a code for estimating NBI heating of fusion plasmas on the Grid. Proc. First EELA-2 Conf, 227-234 (2009)

[7] E. Huedo, R.S. Montero and I.M. Llorente. The GridWay Framework for Adaptive Scheduling and Execution on Grids. J. Scalable Comp.–Prac. Exp. **6** (3), 1-8 (2005)

Proceedings of the First EELA-2 Conference
R. Mayo et al. (Eds.)
CIEMAT 2009
© 2009 The authors. All rights reserved

FAFNER2: adaptation of a code for estimating NBI heating of fusion plasmas on the Grid

M. Rodríguez[1], J. Guasp[1,2], F.Castejón[1,2], A.J. Rubio-Montero[1] and R. Mayo[1]

[1]CIEMAT, Avda. Complutense, 22 - 28040 Madrid (Spain)
{manuel.rodriguez,antonio.rubio,rafael.mayo}@ciemat.es
[2] Laboratorio Nacional de Fusión, Avda. Complutense, 22 - 28040 Madrid (Spain)
{guasp,francisco.castejon}@ciemat.es

Abstract

FAFNER2, a 3D code adapted to the TJ-II helical axis stellarator from the original one developed by Lister at Max Planck IPP, simulates by Monte Carlo methods the Neutral Beam Injection (NBI) technology (a key heating method for most of the fusion experiments worldwide). To the date, FAFNER2 has been usually run at CIEMAT by means of a batch mode on a shared memory Cray architecture, but it has also been translated to MPI. From this point of view, FAFNER2 would not only mean a new code ported to the Grid, but it opens a new strategy for the fusion community. Since it deals with the heating method for the fusion devices, it is able to be coupled to ion kinetic transport codes in a way that the output from FAFNER2 will be the input for the latter. Such a complex workflow, the development of which could be done by means of Kepler Project tools, is of interest for both fusion and grid communities. Into this framework, the FAFNER2 porting process to the grid is the first step of a more ambitious work. In this paper, all these preliminary advances will be described. The steps of the transformation from a SHMEM over MIPS to a Grid over X86 technology will be fully explained. Also, performance and portability gains obtained on each step of the process will be evaluated.

1. Introduction

Nuclear fusion aims to be the next generation of energy source. By employing a mixture of deuterium and tritium (two isotopes of hydrogen) under extreme temperature and pressure, it is possible to reproduce the same reaction that happens inside the starts, thus producing a huge amount of energy.

The main problem of nuclear fusion is a technological one: we still are not able to build a reactor that can start and maintain this kind of reaction for a long time. On the opposite, fusion reactions only last for a short time, and, as Plasma is confined magnetically -employing external coils-, current reactors employ a lot of energy.

To overcome this problem, computer simulation is employed. There are software applications that simulate different sections of a nuclear reactor, obtaining very accurate results. This way, we can study the expected behaviour of the reactor, thus reducing physical experiments. The only drawback is that this kind of software needs a lot of computing resources to produce valid results in a reasonable time period.

In this context, computational grids represent a powerful tool for the fusion community, as its members are able to share their equipment and obtain a big pool of computational resources. Several international projects like EGEE [1], EELA-2 [2] or EUFORIA [3] provide the physical resources -both CPU time and storage- needed for efficient executions of fusion applications. EGEE, for example, provides 45 nodes with more than 15.000 CPUS through its fusion Virtual Organization. Grid computing has already shown its capability to be used to solve several types of problems that appear in fusion plasma physics and that involve distributed calculation: Monte Carlo codes and parameter scan problems are the most suitable for this purpose, as they require high computation time and the problem can be divided in fully independent tasks, which are distributed among remote grid resources.

The aim of this paper is to explain how FAFNER2, an application that simulates NBI (Neutral Beam Injection) heating, was ported from a local cluster to the Grid, in order to take advantage of the aforementioned computational resources. NBI heating is a key factor on the development of fusion reactors. In fact, this technique will be employed at ITER, the next generation fusion reactor, where FAFNER2 is expected to have a relevant role in its design and simulation.

FAFNER original code was released by G.C. Lister, from Max-Planck institute, in 1985 [4]. FAFNER2, a revision of this software, was released one year later. The version employed in this paper is an adaptation of FAFNER2 to the geometry of the TJ-II stellerator located at CIEMAT. It was developed by fusion department of CIEMAT, in collaboration with the original authors [5].

This paper is organized as follows: section 2 explains the physical problem to be solved, as well as a high level description of the code; sections 3 and 4 detail the porting process and analyze its performance; at last, in sections 5 and 6 we discuss our experience and explain our future work.

2. The code and its Physics

The physical problem to solve is to model the neutral beam injection into three-dimensional toroidal plasmas, and to calculate the trajectory of the resultant fast ions until they get lost or are absorbed by the system.

The first part of the code simulates the injection of neutral particles into the torus. As a result, the coordinates and speed of these particles are obtained.

After that, in the second part of the code, this information is used to determinate the initial ionization of these atoms inside the plasma, and then, the energy distribution of the plasma as a result of its interaction with the fast ions. FAFNER2 also computes the energy lost as a result of the collisions between the neutral beam and the scrappers and ducts.

From a mathematical point of view, Monte Carlo techniques are employed to compute the coordinates of a set of fast ions, corresponding to a neutral beam, as well as the appropriate plasma parameters.

The trajectory of these ions and its interaction with the plasma are described in terms of a Fokker-Plank equation. The trajectories of the fast ions are solved with a Boozer flux coordinate system. These numerical techniques are well known and have been previously employed to describe heating of neutral beams both in tokamaks and stellarators fusion devices.

FAFNER2 provides two sets of data on each execution. First, the position and speed of a given number of particles is determined. Second, it obtains different statistics about plasma state: fraction of ionized particles, absolute power provided to the torus, and fraction of lost particles.

3. Implementation

The process of porting FAFNER2 to the Grid was accomplished in three steps.

The first one consisted of a transformation of the message passing mechanism, from SHMEM to MPI, inside a shared memory SGI computer. Then, we ported the application to X86 architecture. And at last, we executed this code on the Grid.

3.1. Message passing mechanism transformation: from SHMEM to MPI

As said, the first task was to change the message passing mechanism, from SHMEM to MPI. SHMEM is a message passing technology present in some SGI systems, but not on standard clusters. As our final goal is running FAFNER2 on the Grid, it is mandatory to use standard technologies as MPI one is.

SHMEM library is based on MPI version 1, and in many cases their routines are equivalent. Thus, the first step was to find and replace these routines. Then, the application was optimized in order to avoid unnecessary inter-thread communications and barriers, and to take advantage of all the possibilities that MPI provides. As a consequence, execution time was slightly reduced. The results are shown in section 4.2.

As Input/Output is different on SHMEM and MPI, it was also necessary to adapt the code in order to have exactly the same output files. These files are employed as entry files in other applications, and any difference will make them miss-work.

3.2. Architecture improvement: from MIPS to X86

After porting the code to MPI, we migrated the application from its obsolete MIPs architecture to the X86 platform.

The first step consisted of deleting obsolete and/or unnecessary packages, replacing them with newer versions. We also minimized the size of the executable by removing all the unused routines from the source code of these packages. In the case of NAG mathematical library, its optimization produced a size reduction of 99%.

Also, a small number of routines utilized in FAFNER2 do not belong to Fortran standard, but to a private library implemented for a certain architecture and operating system, present on the computers when the original version of the program was created, back in 1994. When migrating the application, these routines become unavailable or have a not very significant different functionality. Thus, it is necessary to locate these routines and implement them again, to make the exactly work as expected.

Originally, FAFNER2 was designed to be executed on 64 bits processors. Nevertheless, as EGEE hardware and O.S. are 32 bits, it is necessary to make the application work with this word size. We had to keep the original data size both for integer and real variables, in order to obtain, as much as possible, the same data precision. This situation caused an unalignment of some variables, so we re-ordered its declarations, and then forced alignment by employing a specific flag on the compiler. The final results are equivalent with a deviation lower than 1%.

3.3. Porting to the Grid

Given that the basic idea behind the Grid is being able to launch multiple, concurrent executions of the application, it is necessary to divide the problem to be solved, in order to have one execution per section.

As aforementioned, FAFNER2 is a Monte Carlo code, so it was not necessary to modify the algorithm to subdivide the problem. Instead, in the grid porting process we focused on

dividing the input and joining the output data. In our approach, we implemented two wrappers: one that pre-processes the application input data and one that post-processes the output files (see Figure 1).

FAFNER2 produces output data in two different places: along the program execution -only on the first thread- and near the end of the execution, after gathering data from all threads. This information has been processed when it is shown on the screen or stored in a file (for example, if it is a mean, it has already been calculated), so we modified the application in order to get the raw data, before being processed.

To achieve this, we introduced into the code sentences that export to an XLM file the required variables and partial results. This way, besides having a fully functional application, we produce results that are employed on the gridification. These files, together with the program output, constitute the entry of the post-process wrapper. This wrapper reads all the XML files, and together with the information of the output files (also in plain text format) generates a new output file.

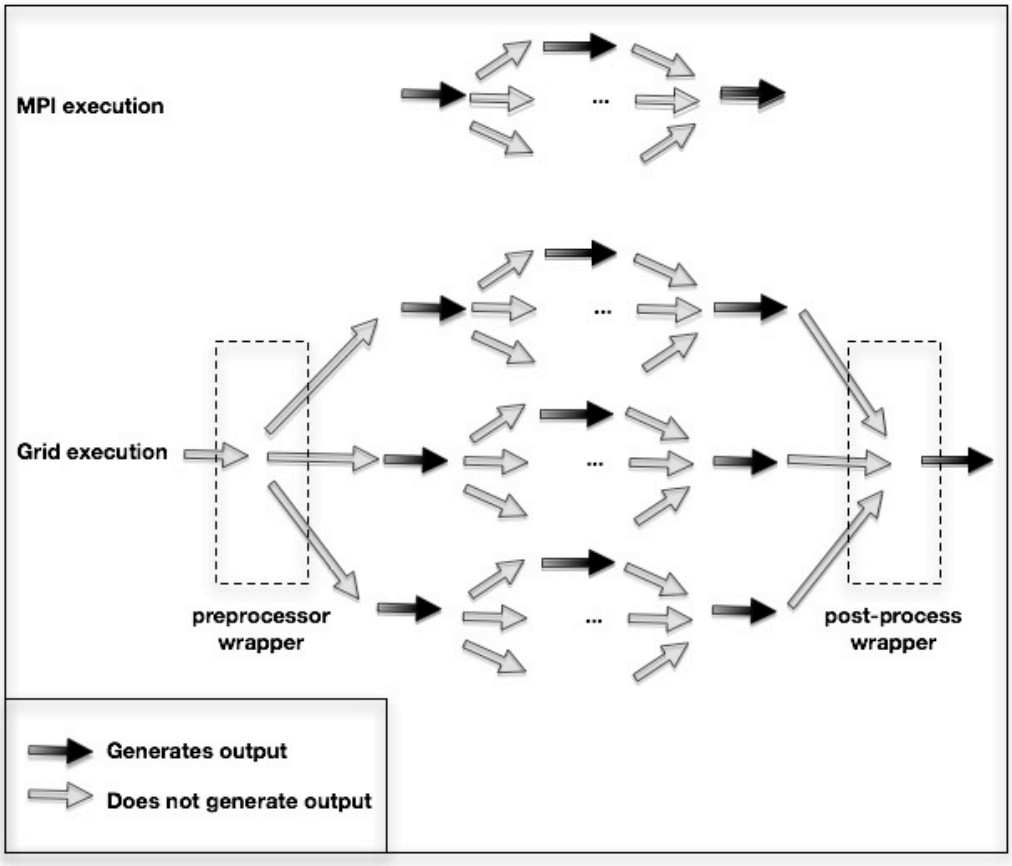

Figure 1. Generation of output files on MPI and Grid versions of FAFNER2

To make the most of the existing Grid infrastructure we decide to employ a metascheduler, in order to manage, control and monitor the execution of our grid application. We have chosen the GridWay metascheduler [6], as it provides some features which are required by FAFNER2:

- Scheduling capabilities: GridWay employs a dynamic scheduling system. It can detect when a new machine has been added or removed from a cluster, and redistribute the work load. Also, it detects the services published by any machine, so the application will only be sent to equipment running the correct MPI version.

- Fault detection & recovery capabilities. Transparently to the end user, GridWay is able to detect and recover from any of the Grid elements failure, outage or saturation conditions [7].

4. Experimental Results

4.1. Testbed Description

The behaviour of the successive FAFNER2 implementations has been analyzed in a testbed based on three resources: one shared memory computer and two MPI-enabled clusters. The main characteristics of these machines are described in Table 1.

Table 1. Characteristics of the testbed resources

Name	Characteristics
Jen50, 122 cores	MIPS R14000@500 MHz, 126 GB total
Lince, 46 nodes (184 slots)	Dual Xeon 3.2 GHz,2 GB each 1Gb/s Ethernet
ce-eela.ciemat.es, 20 nodes (40 slots)	Dual Xeon 3.20 GHz 2 GB each 1Gb/s Ethernet

We also submitted jobs to different remote grid nodes in order to measure file transfer time. We employed three different EGEE nodes, located at different distances of CIEMAT: one at Universidad Complutense de Madrid, one at Universidad Politécnica de Valencia and one at Universidade Federal do Rio de Janerio.

4.2. SHMEM to MPI

First, we studied the performance gain obtained after upgrading the message passing mechanism. Both versions of FAFNER2 were executed 10 times on the same computer under different load conditions, to have realistic results. Figure 2 shows these execution times.

SHMEM usually outperforms MPI on MIPS processors [8]. However, after migrating the application and minimizing its network traffic, we have reduced execution time in 149 seconds, roughly a 10%. Also, time variability has been reduced: the difference between the fastest and slowest execution has been reduced from 458 to 51 seconds.

4.3. SGI to X86

After migrating the application from MIPS to X86, we compared its performance in both platforms. FAFNER2 will be executed on a local cluster when the phenomena to be simulated will calculate a low number of particles (some thousands), so this is a key aspect of the job.

Due to system use policy, MPI applications executed on Lince cluster are limited to 10 slots, so results shown on Figure 3 are, in fact, impressive.

4.4. Cluster to Grid

After porting the application to the grid, we measured its execution time. This allowed us to check, besides the execution time difference, the scalability of the problem. Grid execution

is not only intended to reduce execution time of jobs, but to allow simulating more complicated phenomena, that cannot be coped with local infrastructures.

Measuring Grid performance is complicated. Depending on the chosen parameters, sites and moment on which the application will be run, results can be tremendously variables. To minimize this problem, we only employed an EELA grid node at CIEMAT. This way, we also had the possibility of monitoring system load, and time executions under different conditions -as we did on the previous cases-.

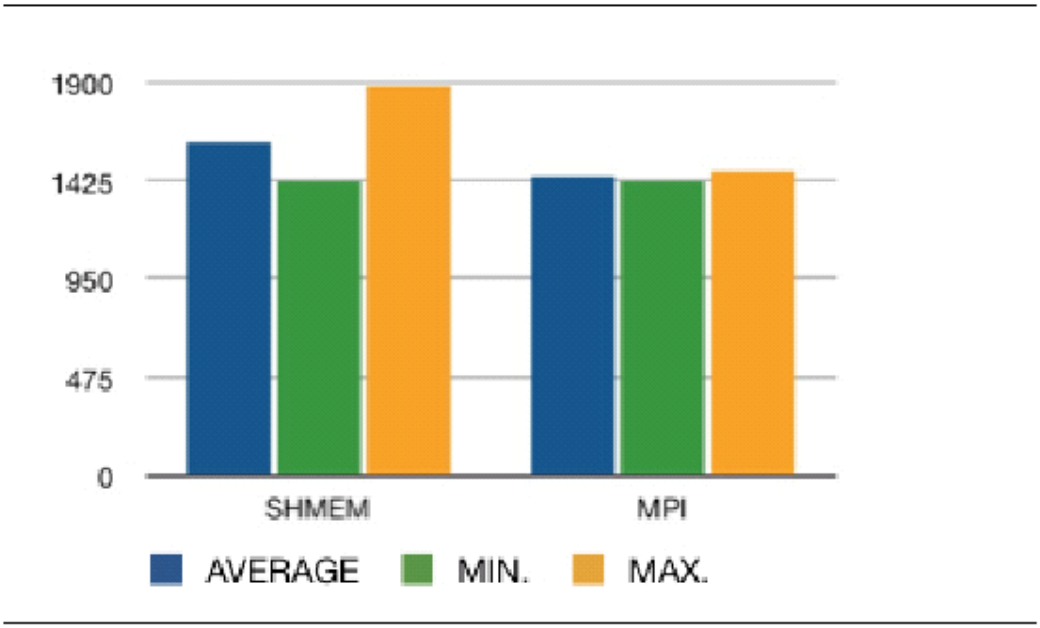

Figure 2. FAFNER2 execution time of SHMEM and MPI versions in Jen50

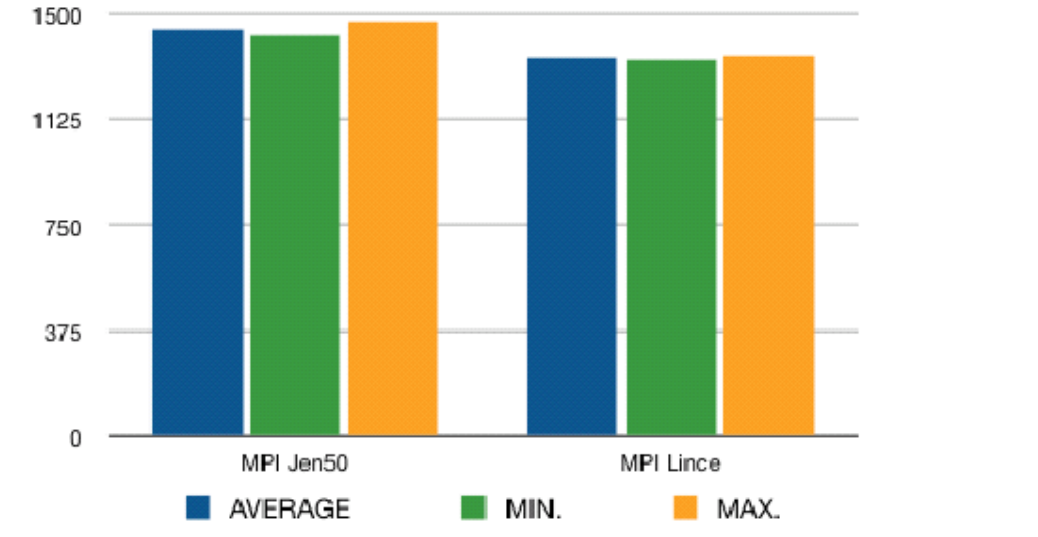

Figure 3. FAFNER2 execution time on Jen50 (32 threads) and Lince (10 threads).

In this case, we need to time not only the application execution, but also the wrappers execution time. These are the obtained results:

- Pre-process wrapper is relatively simple, and its execution very fast. Execution time is less than a second.

- Execution time of FAFNER2 was measured on the EELA Grid node of CIEMAT, called ce-eela.ciemat.es. Obtained results are shown on Figure 4. we kindly ask reader to bear in mind that for Acceptable Use Policy reasons, this figure don't show a real parametric comparison because we couldn't perform the executions on the same number of nodes; thus, we could count on 32 nodes for the *ce-eela.ciemat* case, but only 10 for the *Lince* one.
- Post-process wrapper scales linearly with the number of executions, at about 2 seconds/execution. In this case, with 10 executions, it took about 20 seconds to generate the final result.

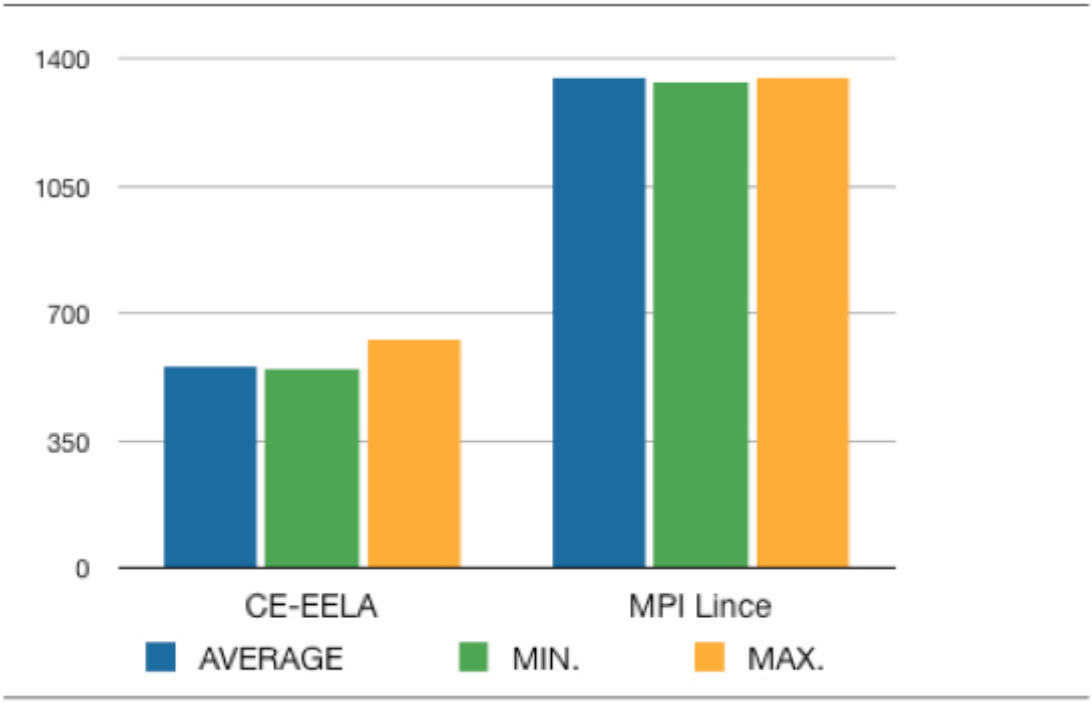

Figure 4. FAFNER2 execution time on Lince and ce-eela.ciemat.es

5. Future work

FAFNER2 is an application that will evolve on the following years. Besides being employed at CIEMAT to simulate the processes inside TJ-II, both on Grid and on MPI clusters, it is expected to simulate NBI heating at ITER [5], the biggest fusion project in the history of mankind. To perform this task, FAFNER2 code must be adapted, providing a clear and easy-to-use interface. As a consequence, the geometry of the sector being simulated will be a different module in the code, so the user will be able to upload it as required.

In order to obtain a greater portability, MPI will be removed from the code, thus making FAFNER2 work on any Grid site. This way, the number of machines in which FAFNER2 can run will be greatly increased. The drawback is the need of implementing more complex pre-processor and post-processor wrappers.

At the same time, it is necessary to analyze FAFNER2 information flow and minimize it, in order to reduce the data being transmitted on the internet. This way execution time is reduced, and the exposure to a data transmission fault is minimized. To accomplish this task, it is necessary to implement a workflow, studying which data to send where and when.

When all this tasks are accomplished, a web interface to manage FAFNER2 and analyze its results will be created, in order to simplify the usage of the application. This interface will

also allow coupling FAFNER2 to other fusion codes such as ISDEP, creating complex workflows that simulate a wide number of plasma phenomena.

6. Conclusions

During this work we have accomplished three fundamental objectives: a change on the message passing paradigm, from SHMEM to MPI; updating the application to a newer architecture, from MIPS to X86; and porting the application to the grid.

The first task, changing the passing message interface, although being a long task was successfully solved, making the application work correctly.

The second task (porting the application from MIPS from X86) led to deep changes in the code in order to obtain the same results. Anyway, we managed to reduce execution time to about one third, maintaining data precision and the same output format.

At last, we checked that FAFNER2 correctly works on a Grid. We developed pre-process and post-process wrappers. Then, we measured the performance of the application, and demonstrated its scalability.

References

[1] http://public.eu-egee.org.

[2] http://www.eu-eela.eu.

[3] http://www.euforia-project.eu.

[4] G. Lister. FAFNER: A Fully 3-D Neutral Beam Injection Code Using Monte Carlo Methods. Max-Planck-Institut für Plasmaphysik, 1985.

[5] J.A. Teubel. Monte Carlo simulations of NBI into the TJ-II helical axis stellarator. Report IPP 4/268, 264(4), 1994.

[6] E. Huedo, R.S. Montero and I.M. Llorente. J. Scalable Comp.–Prac. Exp. **6** (3), 1-8 (2005)

[7] E. Huedo, R. S. Montero, and I. M. Llorente. Evaluating the Reliability of Computational Grids from the End User's Point of View. Journal of Systems Architecture, 52(12):727–736, Dec. 2006.

[8] G.R. Luecke, S. Spanoyannis and M. Kraeva. The performance and scalability of SHMEM and MPI-2 one-sided routines on a SGI Origin 2000 and a Cray T3E-600. Concurrency Computat.: Pract. Exper. **16**, 1037-1060 (2004)

Teh Internets is a series of tubes transmittin LOLcats